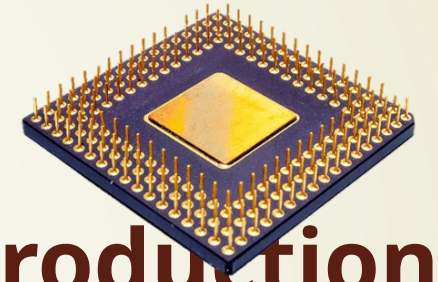


Introduction aux systèmes d'exploitation et à leur fonctionnement

2022 – 2023

JFA - 1

DUT Informatique – Semestre 1

Ressource R1.04

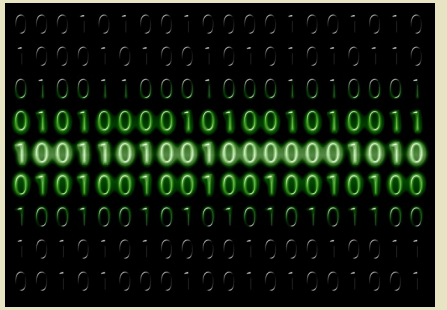
Responsable : Jean-François ANNE

jean-francois.anne@unicaen.fr

http://www.jfanne.fr

Sommaire

- Caractéristiques et types de systèmes d'exploitations
- Langage de commande (commandes de base, introduction à la programmation des scripts)
- Gestion des processus (création, destruction, suivi, etc.)
- Gestion des fichiers (types, droits, etc.)
- Gestion des utilisateurs (caractéristiques, création, suppression, etc.)
- Principes de l'installation et de la configuration d'un système : notion de noyau, de pilotes, de fichiers de configuration, boot système…



Présentation des systèmes d’exploitation

JFA - 3

DUT Informatique – Semestre 1

Ressource R1.04

Responsable : Jean-François ANNE

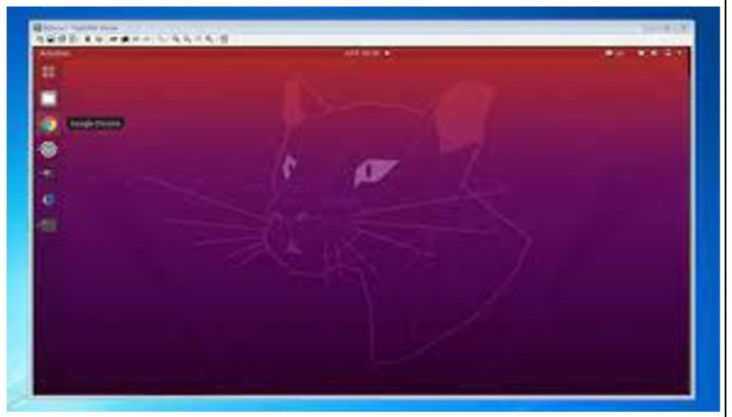
I. Présentation des systèmes d’exploitation :

Tout au long de cette ressource, nous apprendrons comment fonctionnent les systèmes d'exploitation. Mais, avant d’approfondir nos connaissances de Linux et Windows, deux des plus fascinants systèmes d’exploitation, nous allons définir ce qu’est un système d’exploitation et en voir les composantes.

A. Qu'est-ce qu'un système d'exploitation (operating system – OS) ? :

Vous en connaissez déjà plusieurs :

Et il y a eu la guerre entre Microsoft et Linux et maintenant la guerre entre Microsoft et Google à propos de leur OS respectif...

	
Apple Mac OS X Leopard	Microsoft Windows 10
	
Linux – Ubuntu	Chrome OS

B. Introduction :

- Il existe deux catégories de logiciels :
  - Les programmes systèmes pour le fonctionnement des ordinateurs,
  - Les programmes d'application qui résolvent les problèmes des utilisateurs.

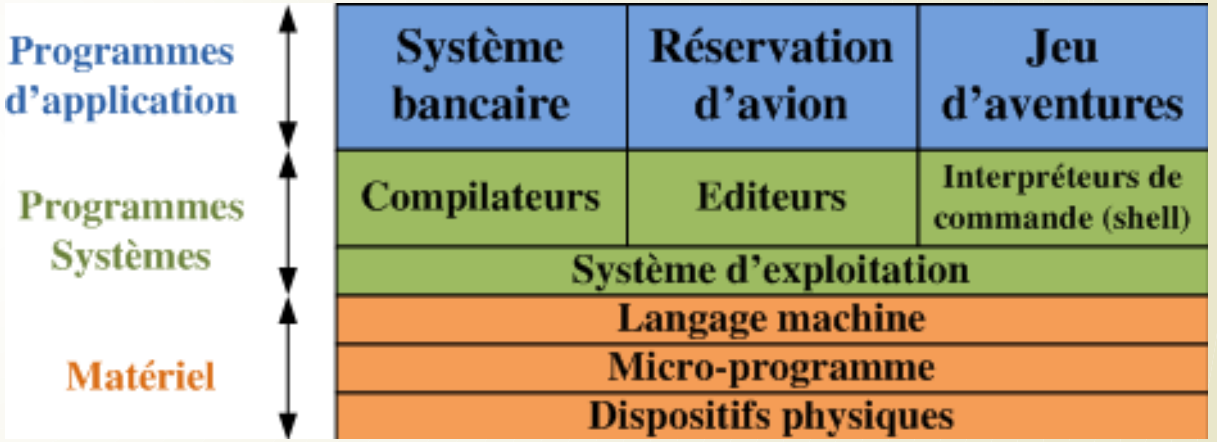
Le programme « système d'exploitation » est le programme fondamental des programmes systèmes. Il contrôle les ressources de l'ordinateur et fournit la base sur laquelle seront construits les programmes d'application.

- Deux modes de fonctionnement :
  - Le mode noyau ou superviseur et
  - Le mode utilisateur (compilateur, éditeur, programmes utilisateurs ...).

Un ordinateur « nu »

- Est une machine inutilisable, car :
  - Programmation en langage binaire seulement
  - Accès aux périphériques très difficiles
  - Exécution d'un seul programme à la fois
- Pour exécuter un programme, il faut :
  - Aller le chercher sur le disque dur :
    - Trouver sa position
    - Lire les mots qui le décrivent
  - Le mettre en mémoire
    - • Lui allouer un espace
  - L'exécuter…
    - • Gestion du clavier par ce programme?
    - • Gestion de l'écran ?

Un ordinateur contient…



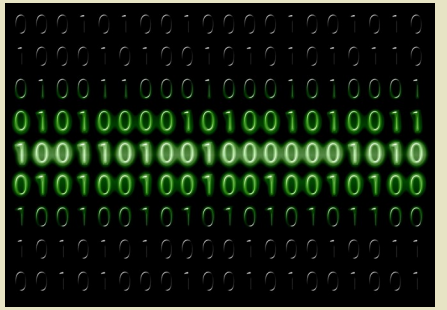
- Des dispositifs physiques : ils se composent de circuits intégrés, de fils électriques, de périphériques physiques …
- Un microprogramme : c'est un logiciel de contrôle des périphériques (interprète).
- Il utilise le langage machine : C'est un ensemble (entre 50 et 300) d'instructions élémentaires (ADD, MOVE, JUMP) pour effectuer le déplacement des données, des calculs, ou la comparaison de valeurs.
- Système d'exploitation propose un ensemble d'instructions plus simples, comme LIRE UN BLOC DU FICHIER.

L’ordinateur :

- Un ordinateur contient :
  - Un ou plusieurs processeurs,
  - Une mémoire principale,
  - Des horloges,
  - Des terminaux,
  - Des disques,
  - Des interfaces de connexion à des réseaux et
  - Des périphériques d'entrées/sorties.

La complexité évidente du matériel implique la réalisation d'une machine virtuelle qui gère le matériel :

c'est le système d'exploitation.



Caractéristiques et types des systèmes d'exploitation

JFA - 10

DUT Informatique – Semestre 1

Ressource R1.04

Responsable : Jean-François ANNE

Un système d’exploitation

Système d’Exploitation (SE) = Operating System (OS)

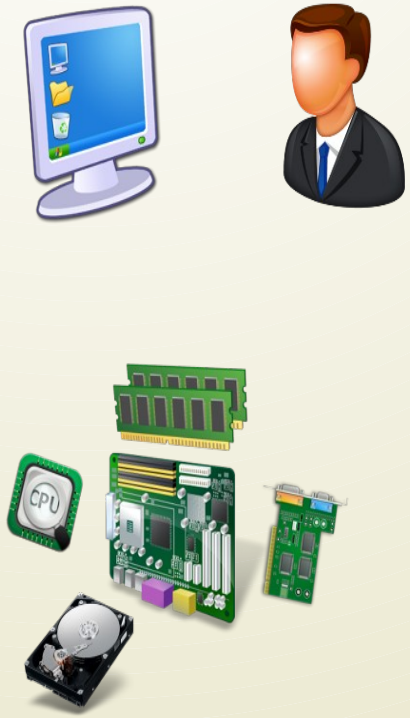
Le SE est le logiciel de base indispensable à tout système informatique !

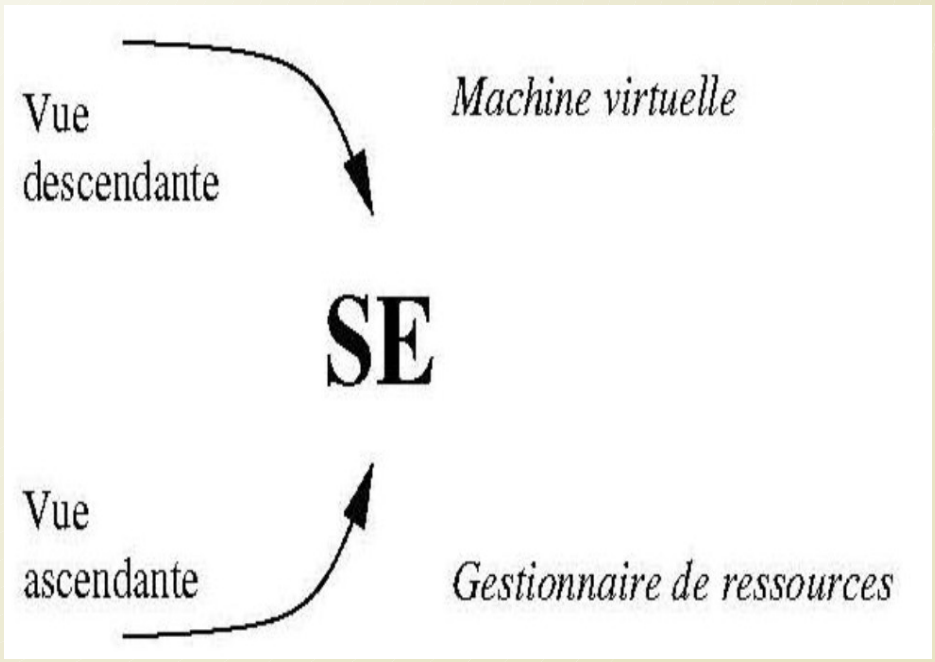
Le SE fonctionne comme intermédiaire entre l’utilisateur et le système informatique

C. Définitions

Il a 2 fonctions :

- **De présenter une machine virtuelle :** Son rôle est de masquer les éléments fastidieux liés au matériel et permettre à l'utilisateur une exploitation simple et efficace de la machine ;
- **D'être un gestionnaire de ressources** : Gérer l'ordonnancement et le contrôle de l'allocation des processeurs, des mémoires et des périphériques d'E/ S entre les différents programmes qui y font appel.

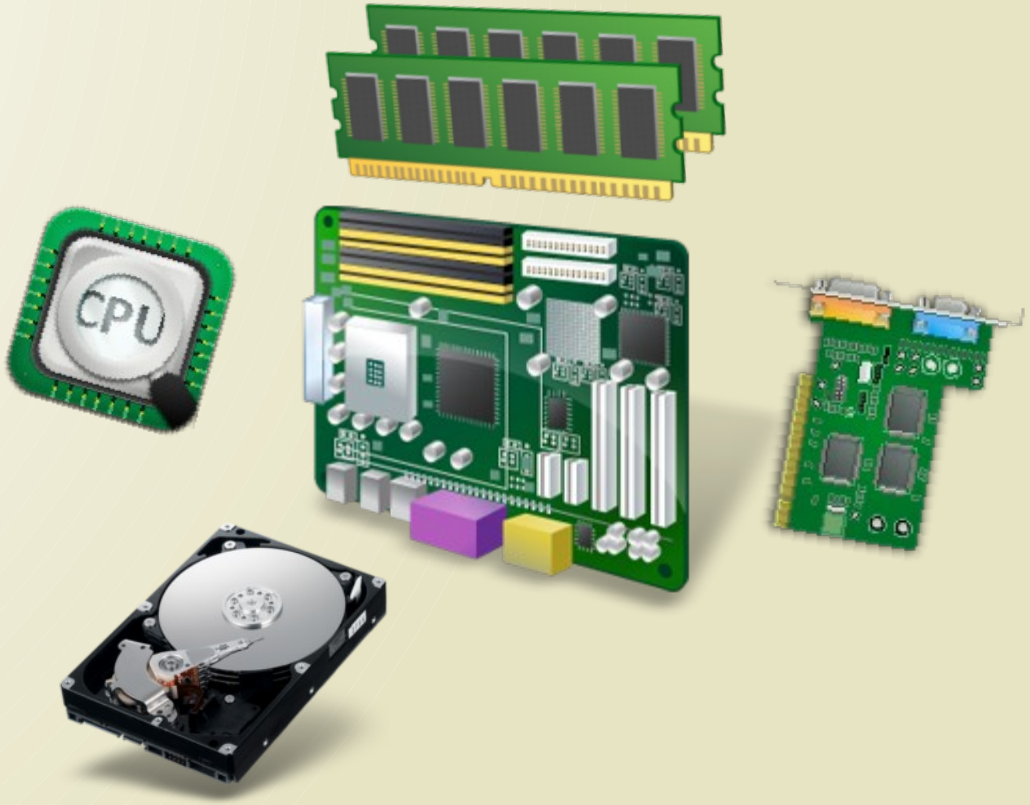




Vue du système d'exploitation

Vue Ascendante :

- Gestionnaire des ressources :
  - Mémoire
  - Processeur
  - Périphériques
  - Fichiers
  - Stockage
  - …



Vue Descendante :

- Machine virtuelle :
  - Masquer les éléments fastidieux (logiciels et matériels), Processeur
  - Permettre à l'utilisateur une exploitation simple et efficace de la machine
  - …

Machine étendue ou machine virtuelle.

- **Être une machine virtuelle** signifie transformer un assemblage de chips et de circuits en un appareil plus utilisable. C'est-à-dire de travailler avec un outil moderne qui offre une abstraction simple :
  - au niveau des entrées/sorties,
  - de l'utilisation de la mémoire,
  - de la gestion des fichiers,
  - de la protection et du contrôle des erreurs,
  - de l'interaction des programmes entre eux et de leur contrôle.

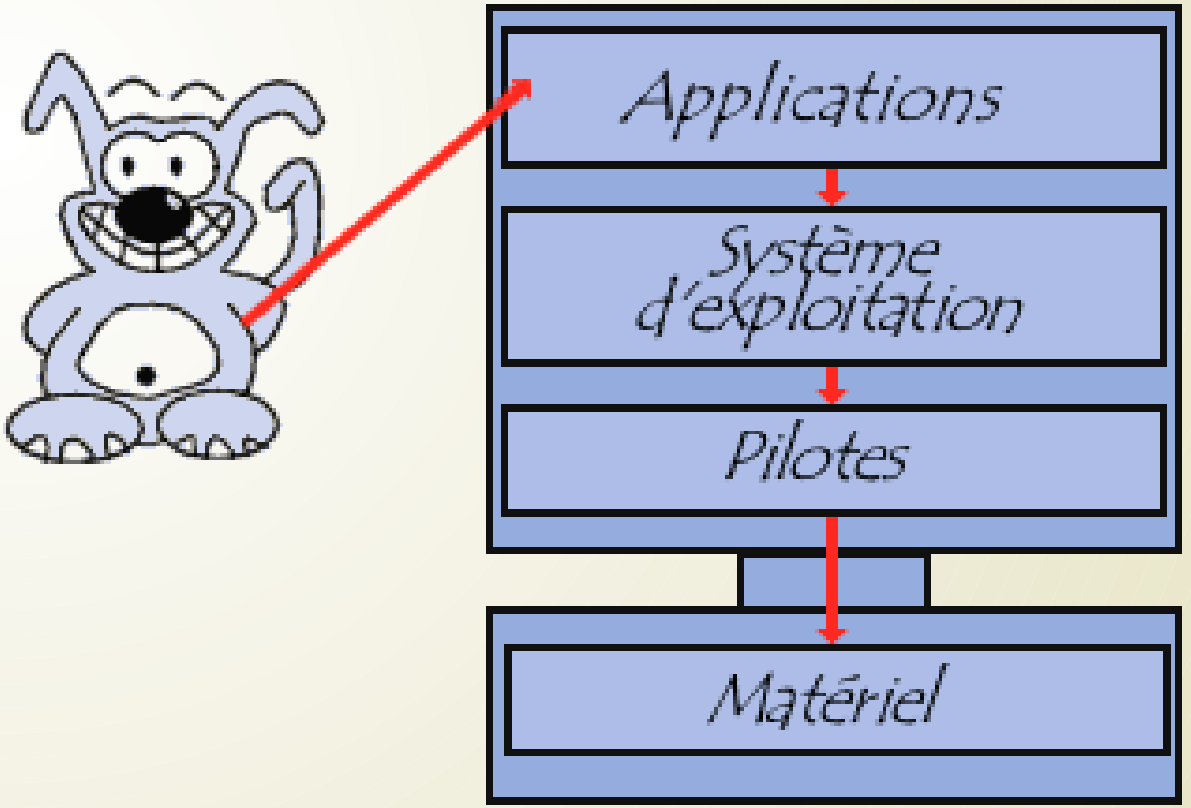
- En deux mots : d'éviter au programmeur de connaître les détails électroniques de tel ou tels chips et de permettre à l'utilisateur de sauvegarder ses fichiers sans se soucier du type de modulation de fréquence utilisée pour stocker les informations.
  - Son rôle est donc de masquer des éléments fastidieux liés au matériel, comme les interruptions, les horloges, la gestion de la mémoire, la gestion des périphériques (déplacement du bras du lecteur de disquette) ...
- Ex. **READ** et **WRITE** => 13 paramètres sur 9 octets ; en retour le contrôleur renvoie 23 champs d'état et d'erreurs regroupés sur 7 octets.

Gestionnaire de ressources.

- Un ordinateur se compose de ressources (périphériques, mémoires, terminaux, disques ...).
- Donc l'autre fonction du système d'exploitation, **c'est le partage des ressources**. Il se fait entre les programmes appelés plus justement les processus. Ce rôle de policier du système d'exploitation permet d'éviter les conflits d'utilisation de la mémoire, des périphériques d'entrées/sorties, des interfaces réseau... etc. On peut facilement imaginer ce qui arriverait si trois programmes essayaient d'imprimer en même temps sans qu'un certain ordre ne soit respecté. Ce travail d'alternance assuré par le système d'exploitation permet de mettre de l'ordre dans un chaos potentiel.
- De plus, lorsque l'ordinateur est utilisé par plusieurs usagers (presque tout le temps), le partage de la mémoire et surtout sa protection demeure une priorité absolue. **En tout temps, un bon système d'exploitation connaît l'utilisateur d'une ressource, ses droits d'accès, son niveau de priorité.**

- Le système d'exploitation permet l'ordonnancement et le contrôle de l'allocation des processeurs, des mémoires et des périphériques d'E/S entre les différents programmes et utilisateurs qui y font appel.
- Par exemple 3 programmes essaient d'imprimer simultanément leurs résultats sur une même imprimante :
  - => recours à un fichier tampon sur disque.
- Autre exemple, l'accès simultané à une donnée ; lecture et écriture concurrentes (par deux processus) sur un même compteur.
- *Ce rôle de gestionnaire de ressources est crucial pour les systèmes d'exploitation manipulant plusieurs tâches en même temps (multi-tâche).*

D. Organisation d'un système d'exploitation



Les différentes couches logicielles qui constituent un ordinateur

- Le système d'exploitation est le programme fondamental des logiciels *système*: il contrôle les ressources de l’ordinateur et fournit la base essentielle sur laquelle sont construire les logiciels *application*. Le système d'exploitation opère sur un ordinateur constitué d'un ou plusieurs processeurs, d'une mémoire principale, d'horloges, de disques, de périphériques E/S, d'interfaces de connexion à des réseaux... etc. C'est la complexité toujours croissante du matériel qui a entraîné au cours des années, un développement rapide et spectaculaire des systèmes d'exploitation. Aujourd’hui, on parle de virtualité à tous égards ce qui permet aux usagers et aux programmeurs (de plus en plus) d'utiliser l'ordinateur avec un niveau d’abstraction élevé.

- Le **système d'exploitation** est chargé d'assurer la liaison entre les ressources matérielles, l'utilisateur et les applications (traitement de texte, jeu vidéo, ...). Ainsi lorsqu'un programme désire accéder à une ressource matérielle, il ne lui est pas nécessaire d'envoyer des informations spécifiques au périphérique, il lui suffit d'envoyer les informations au système d'exploitation, qui se charge de les transmettre au périphérique concerné via son pilote. En l'absence de pilotes il faudrait que chaque programme reconnaisse et prenne en compte la communication avec chaque type de périphérique ! Il permet ainsi de "dissocier" les programmes et le matériel, afin notamment de simplifier la gestion des ressources et offrir à l'utilisateur une interface homme-machine (notée «IHM») simplifiée afin de lui permettre de s'affranchir de la complexité de la machine physique.

Tous les systèmes d’exploitation sont donc segmentés en couches pour permettre un meilleur contrôle de l’ensemble de l'ordinateur :

Les parties d’un système Unix.

Rôles du système d'exploitation .

- Transformer une machine matérielle en une machine utilisable, c'est-à-dire fournir des outils adaptés aux besoins indépendamment des caractéristiques physiques,
- Optimiser l'utilisation du matériel principalement pour des raisons économiques.

- Mais il faut également la garantie d'un bon niveau en matière de :
  - Sécurité : intégrité, contrôle des accès confidentialité…
  - Fiabilité : degré de satisfaction des utilisateurs même dans des conditions hostiles et imprévues,
  - Efficacité : performances et optimisation du système pour éviter tout surcoût en termes de temps et de places consommées par le système au détriment de l'application.

Qualités requises du système d’exploitation.

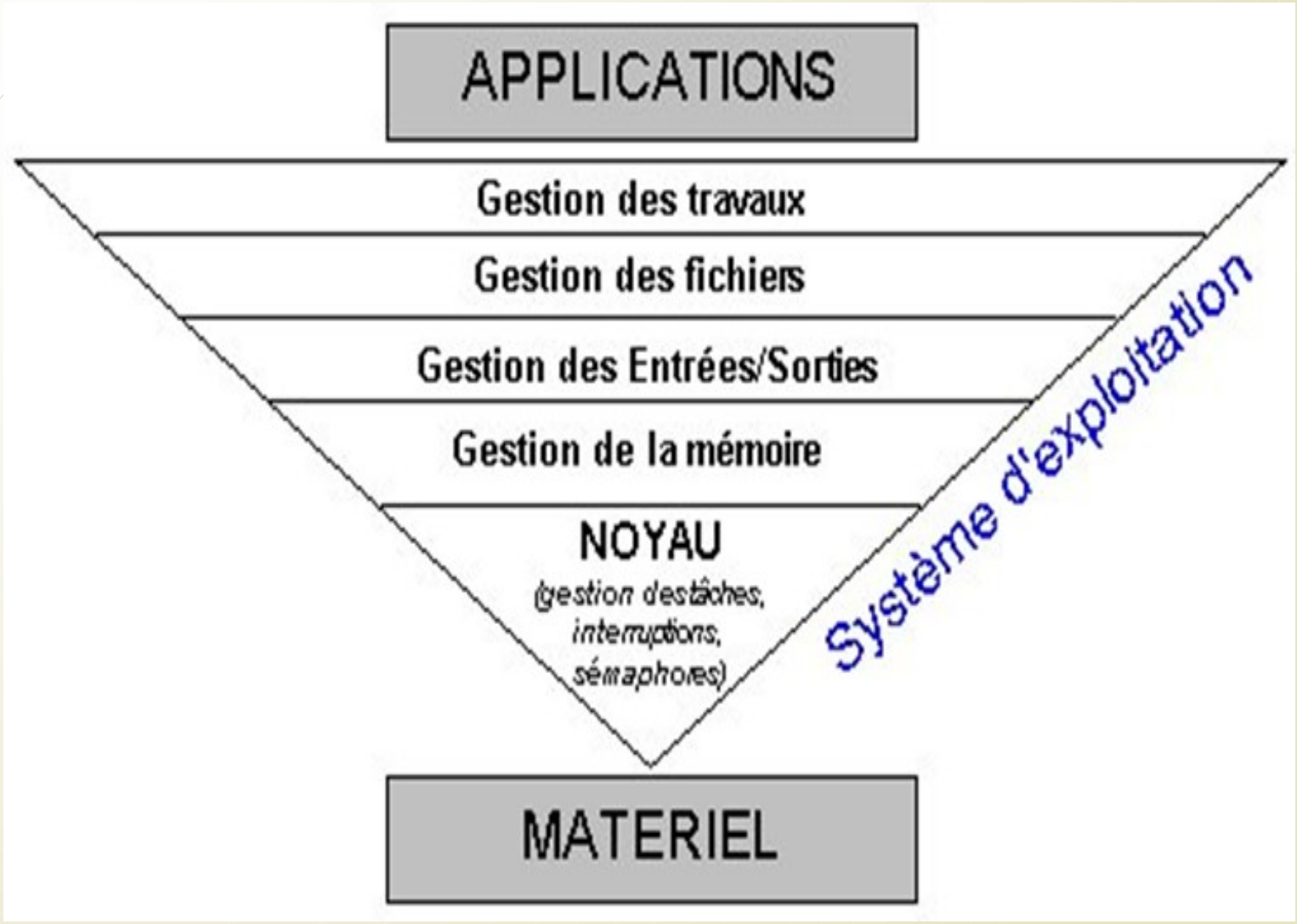
Un système d’exploitation doit se « faire oublier » : la fonction d’un ordinateur est d’exécuter les applications, pas le système d’exploitation.

- Utilisation efficace des ressources
- Fiabilité
- Tolérance aux fautes (du matériel, des utilisateurs, des programmes)
- La qualité de l’interface (en particulier pour les systèmes interactifs)
- Convivialité
- Simplicité d’utilisation
- Documentation
- Bonne intégration au réseau
- Sécurité et protection
- Répertoire étendu des fonctions

Les rôles du système d'exploitation sont divers :

- **Gestion du processeur** : le système d'exploitation est chargé de gérer l'allocation du processeur entre les différents programmes grâce à un **algorithme d'ordonnancement**. Le type d'ordonnanceur est totalement dépendant du système d'exploitation, en fonction de l'objectif visé.
- **Gestion de la mémoire vive** : le système d'exploitation est chargé de gérer l'espace mémoire alloué à chaque application et, le cas échéant, à chaque usager. En cas d'insuffisance de mémoire physique, le système d'exploitation peut créer une zone mémoire sur le disque dur, appelée « **mémoire virtuelle** ». La mémoire virtuelle permet de faire fonctionner des applications nécessitant plus de mémoire qu'il n'y a de mémoire vive disponible sur le système. En contrepartie cette mémoire est beaucoup plus lente.
- **Gestion des entrées/sorties** : le système d'exploitation permet d'unifier et de contrôler l'accès des programmes aux ressources matérielles par l'intermédiaire des pilotes (appelés également gestionnaires de périphériques ou gestionnaires d'entrée/sortie).

- **Gestion de l'exécution des applications** : le système d'exploitation est chargé de la bonne exécution des applications en leur affectant les ressources nécessaires à leur bon fonctionnement. Il permet à ce titre de « tuer » une application ne répondant plus correctement.
- **Gestion des droits** : le système d'exploitation est chargé de la sécurité liée à l'exécution des programmes en garantissant que les ressources ne sont utilisées que par les programmes et utilisateurs possédant les droits adéquats.
- **Gestion des fichiers** : le système d'exploitation gère la lecture et l'écriture dans le système de fichiers et les droits d'accès aux fichiers par les utilisateurs et les applications.
- **Gestion des informations** : le système d'exploitation fournit un certain nombre d'indicateurs permettant de diagnostiquer le bon fonctionnement de la machine.


APPLICATIONS
Gestion des travaux
Gestion des fichiers
Gestion des Entrées/Sorties
Gestion de la mémoire
NOYAU
(gestion des tâches,
interruptions,
sémaphores)
Système d'exploitation
MATERIEL

Composantes du système d'exploitation

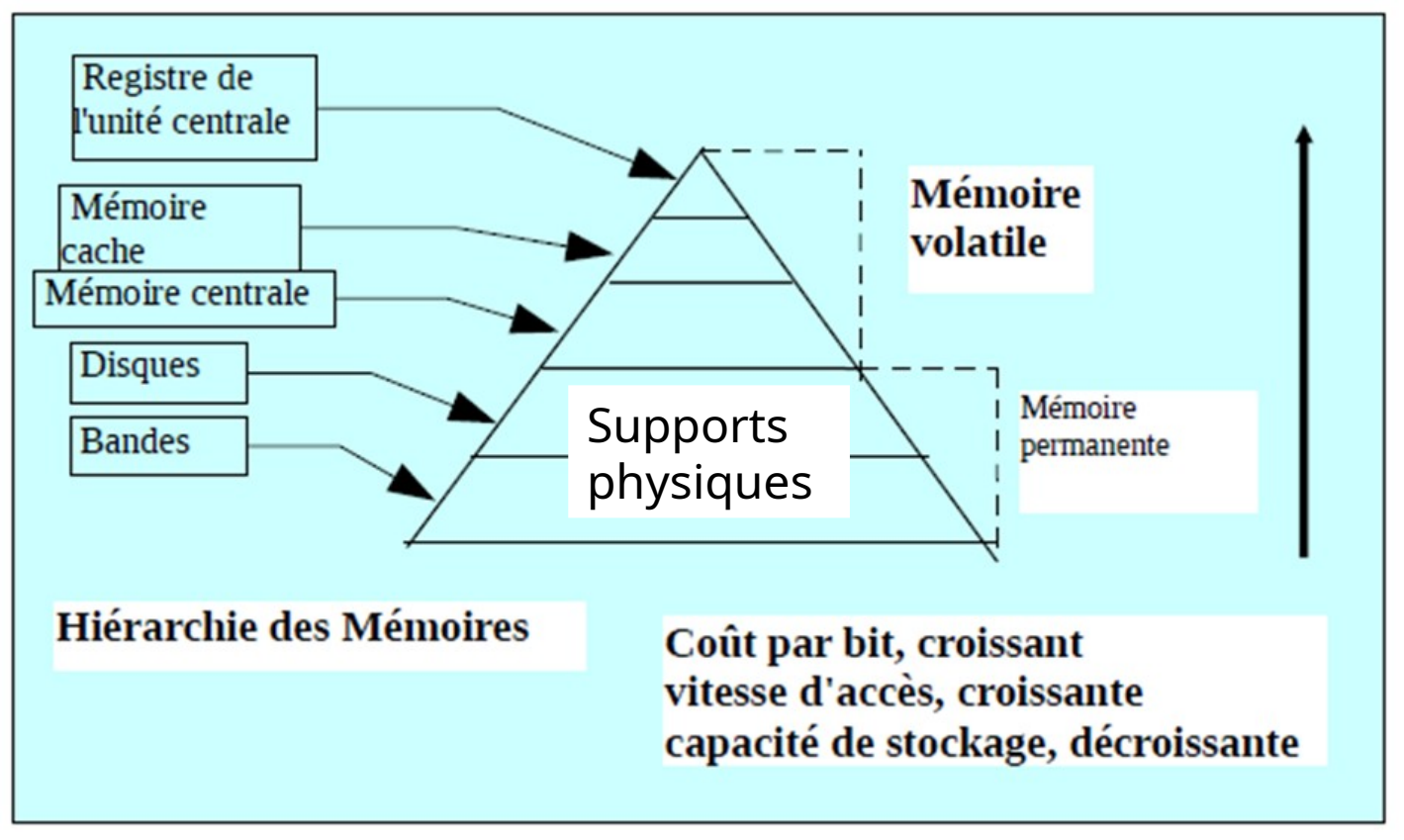
- Le système d'exploitation est composé d'un ensemble de logiciels permettant de gérer les interactions avec le matériel. Parmi cet ensemble de logiciels on distingue généralement les éléments suivants :
  - Le **noyau** (en anglais **kernel**) représentant les fonctions fondamentales du système d'exploitation telles que la gestion de la mémoire, des processus, des fichiers, des entrées-sorties principales, et des fonctionnalités de communication.
  - L'**interpréteur de commande** (en anglais **Shell**, traduisez « *coquille* » par opposition au noyau) permettant la communication avec le système d'exploitation par l'intermédiaire d'un langage de commandes, afin de permettre à l'utilisateur de piloter les périphériques en ignorant tout des caractéristiques du matériel qu'il utilise, de la gestion des adresses physiques, etc.
  - Le **système de fichiers** (en anglais « *file system* », noté *FS*), permettant d'enregistrer les fichiers dans une arborescence.

Plusieurs fonctionnalités de gestion

- Du processeur : allocation du processeur aux différents programmes.
- Des objets externes : principalement les fichiers.
- Des entrées-sorties : accès aux périphériques, via les pilotes.
- De la mémoire : segmentation et pagination.
- De la concurrence : synchronisation pour l'accès à des ressources partagées.
- De la protection : respect des droits d'accès aux ressources.
- Des accès au réseau : échange de données entre des machines distantes.

Les supports physiques du système d'exploitation

- Le système d'exploitation doit pouvoir stocker à long terme de grandes quantités d'informations : ses propres modules, les programmes d'application, les bibliothèques de fonctions, les programmes et les données des utilisateurs.



- Le support physique (mémoire secondaire) doit être de grande capacité et à un faible coût. Par contre il possède des temps d'accès beaucoup plus long que les supports volatiles (mémoire centrale).
- Le système d'exploitation doit assurer la correspondance entre la représentation physique des informations et la représentation logique qu'en a l'utilisateur.
- Durée de vie des supports physiques :

https://www.abavala.com/wp-content/uploads/storage-media-lifespan.jpg

Les différentes classes des systèmes d'exploitation :

Selon les services rendus

- **mono/multi-tâche :**
  - Multi-tâche : capacité du système à pouvoir exécuter plusieurs processus simultanément ; par exemple effectuer une compilation et consulter le fichier source du programme correspondant.
    - *C'est le cas d'Unix, D'OS/2 d'IBM, et de Windows*
- **mono/multi-utilisateurs :**
  - Multi-utilisateur : capacité à pouvoir gérer un panel d'utilisateurs utilisant simultanément les mêmes ressources matérielles.
    - *C'est le cas d'Unix, de MVS, de Gecos, …*

Selon leur architecture

- **Systèmes centralisés :**
  - L'ensemble du système est entièrement présent sur la machine considérée.
  - Les machines éventuellement reliées sont vues comme des entités étrangères disposant elles aussi d'un système centralisé.
  - Le système ne gère que les ressources de la machine sur laquelle il est présent.
  - C'est le cas d'UNIX, même si les applications réseaux (X11, FTP, Mail ...) se sont développées.

Systèmes répartis (distributed systems) :

- Les différentes abstractions du système sont réparties sur un ensemble (domaine) de machines (site).
- Le système d'exploitation réparti apparaît aux yeux de ses utilisateurs comme une machine virtuelle monoprocesseur même lorsque cela n'est pas le cas.
- Avec un système réparti, l'utilisateur n'a pas à se soucier de la localisation des ressources. Quand il lance un programme, il n'a pas à connaître le nom de la machine qui l'exécutera.
- Ils exploitent au mieux les capacités de parallélisme d'un domaine.
- Ils offrent des solutions aux problèmes de la résistance aux pannes.

Selon leur capacité à évoluer

- **Systèmes fermés (ou propriétaires) :**
  - Extensibilité réduite : Quand on veut rajouter des fonctionnalités à un système fermé, il faut remettre en cause sa conception et refaire une archive (système complet).
    - *C'est le cas d'Unix, MS-Dos …*
  - Il n'y a aucun ou peu d'échange possible avec d'autres systèmes de type différent, voir même avec des types identiques.
    - *C'est le cas entre UNIX, BSD et System V.*
- **Systèmes ouverts :**
  - Extensibilité accrue : Il est possible de rajouter des fonctionnalités et des abstractions sans avoir à repenser le système et même sans avoir à l'arrêter sur une machine.
  - Cela implique souvent une conception modulaire basée sur le modèle « client-serveur ».
  - Cela implique aussi une communication entre systèmes, nécessitant des modules spécialisés.

Selon l'architecture matérielle qui les supporte

- **Architecture monoprocesseur (temps partagé ou multi-programmation) :**
  - Ressource processeur unique : Il a fallu développer un mécanisme de gestion des processus pour offrir un (pseudo) parallélisme à l'utilisateur : c'est la multi-programmation ; il s'agit en fait d'une commutation rapide entre les différents processus pour donner l'illusion d'un parallélisme.
- **Architectures multiprocesseurs (parallélisme) :**
  - On trouve une grande variété d'architectures multiprocesseurs :
    - **SIMD** (Single Instruction Multiple Data) : Tous les processeurs exécutent les mêmes instructions mais sur des données différentes.
    - **MIMD** (Multiple Instructions Multiple Data) : Chaque processeur est complètement indépendant des autres et exécute des instructions sur des données différentes.
    - **Pipeline** : Les différentes unités d'exécution sont mises en chaîne et

- On parle aussi d'architecture ***faiblement*** ou ***fortement couplée***.

- ***Architecture fortement couplée*** : Ce sont principalement des architectures à mémoire commune.
- ***Architecture faiblement couplée*** : Ce sont des architectures où chaque processeur possède sa propre mémoire locale ; c'est le cas d'un réseau de stations.
- ***Architecture mixte*** : Ce sont des architectures à différents niveaux de mémoire (commune et privée).

- ***Remarque :***

Il n'y a pas de système universel pour cette multitude d'architectures. Les constructeurs de supercalculateurs ont toujours développé leurs propres systèmes. Aujourd'hui, compte tenu de la complexité croissante des systèmes d'exploitation et du coût inhérent, la tendance est à l'harmonisation notamment via le développement de systèmes polyvalents.

Systèmes temps-réel :

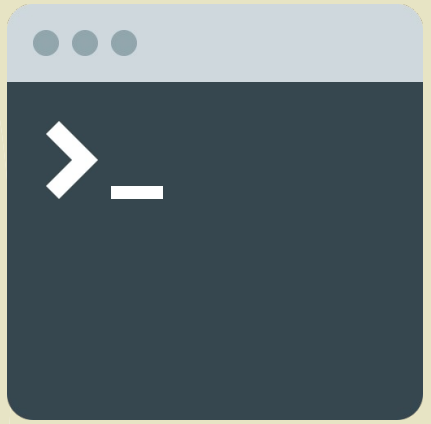
- Ce sont des systèmes pour lesquels l'exécution des programmes est soumise à des contraintes temporelles. Les résultats de l'exécution d'un programme n'est plus valide au delà d'un certain temps connu et déterminé à l'avance.
- Généralement, on trouve des systèmes « temps réel » dans les systèmes embarqués (satellites, sondes, avions, trains, téléphones portables, ...).
- On distingue deux types de contraintes temporelles :
  - Les contraintes strictes et
  - Les contraintes relatives.
- Pour garantir ces contraintes, le système possède des mécanismes spécifiques dont le but est de réduire l'indéterminisme des durées d'exécution des programmes.
- C'est le cas de **Linux-RT**.

Synthèse :

Système	Codage	Mono-Utilisateur	Multi-Utilisateur	Mono-Tâche	Multi-Tâche	Type MT
DOS	8/16 bits	X		X		Mono
Windows 3.1	16/32 bits	X			X	Coopératif
Windows 95/98/Me	32 bits	X			X	Coopératif/ Préemptif
Windows NT/2000	32 bits		X		X	Préemptif
Windows XP/10	32/64 bits		X		X	Préemptif
Unix/LInux	32/64 bits		X		X	Préemptif
MacOs	32 bits		X		X	Préemptif
VMS	32 bits		X		X	Préemptif

- https://fr.wikipedia.org/wiki/Multit%C3%A2che#Multit%C3%A2che_coop%C3%A9ratif

Généralités sur UNIX



JFA - 41

DUT Informatique – Semestre 1

Ressource R1.04

Responsable : Jean-François ANNE

1969 - 1979 : les premiers pas universitaires

- Eté 1969 : Ken Thompson, aux BELL Laboratories, écrit la version expérimentale d'UNIX : système de fichiers exploité dans un environnement mono-utilisateur, multi-tâche, le tout étant écrit en assembleur.
- 1ère justification officielle : traitement de texte pour secrétariat.
- Puis : étude des principes de programmation, de réseaux et de langages.
- En 1972, Dennis Ritchie implémente le langage C, à partir du langage interprété B, écrit par Ken Thompson.
- Eté 1973 : réécriture du noyau et des utilitaires d'UNIX en C.
- En 1974 distribution d'UNIX aux Universités (Berkeley et Columbia notamment). Il se compose alors :
  - d'un système de fichiers modulaire et simple,
  - d'une interface unifiée vers les périphériques par l'intermédiaire du système de fichiers,
  - du multi-tâche
  - et d'un interprète de commandes flexible et interchangeable.

1979 - 1984 : les premiers pas commerciaux

En 1979, avec la version 7, **UNIX se développe commercialement** :

- Par des sociétés privées comme Microport (1985), Xenix-Microsoft (1980) … qui achetèrent les sources et le droit de diffuser des binaires.
- Des UNIX like apparaissent ; le noyau est entièrement réécrit.
- L'université de Berkeley fait un portage sur VAX (UNIX 32V).
- AT&T vend la version 7 sur les ordinateurs de la gamme PDP 11.

1984 - 1993 *… : la standardisation*

En 1984 le Système V.2 est adopté comme standard.

En 1984 X/Open est chargée d'organiser la portabilité d'UNIX.

En 1985 AT&T publie SVID (System V Interface Definition) qui définit l'interface d'application du Système V.2 et non pas son implémentation.

En 1986, le Système V.3 apporte les Streams, les librairies partagées et RFS (Remote File Sharing).

En 1993, X/Open lance le COSE (Common Open Software Environment). Il s'agit d'accords entre constructeurs pour le développement d'applications dans un environnement commun. L'authentification d'UNIX appartient désormais à un consortium de constructeurs (USL, HP, IBM, SUN …).

1991 - ... : LINUX, le renouveau d'UNIX

LINUX est une implantation libre des spécifications POSIX (1003.1) avec des extensions System V (AT&T) et BSD (Berkeley).

En 1991, Linus B. Torvalds (Helsinki) étudie MINIX (A. Tannenbaum)

Août 1991 : 1ère version de LINUX 0.01. C'est une réécriture de MINIX, avec des ajouts de nouvelles fonctionnalités et la diffusion des sources sur « Internet »

-> une version instable

Mars 1994 : 1ère version stable.

Janvier 2004 : La version stable est la 2.6.0 respectent la norme POSIX (code source portable) et le code source est gratuit.

Novembre 2013 : La version stable est la 3.12.0, elle est écrite en C et en assembleur, sous licence GNU GPL 2.

Convention de numérotation des versions Linux : x.y.z :

- x : numéro de version.
- y : si pair, désigne une version stable, sinon désigne une version en Bêta-test.
- z : incrémenté à chaque correction de bug.

Pour connaître la version du noyau en cours :

```
prompt> uname -r
5.15.0-47-generic
prompt>
```

Pour connaître la distribution utilisée :

```
prompt> cat /etc/issue
Ubuntu 22.04.1 LTS
prompt>
```

Une distribution Linux comprend le noyau, les pilotes, les bibliothèques, les utilitaires, ...

- Slackware : la première ...
- Red Hat, Fedora : le concept de paguetage ...
- Debian : non commerciale et de grande qualité ...
- S.uS.E : à l'origine de Slackware ... grande robustesse ...
- Mandrake : Caennaise basée sur RedHat ...
- Caldera : inclut des produits commerciaux ...
- Gentoo : gestionnaire de paquetage *Portage* ...
- Trinux : fonctionne uniquement en mémoire ... outils d'audit des réseaux ...
- TurboLinux : version en cluster, payante, pour gros serveurs ...
- Knoppix : très populaire, sans disque dur ...
- Ubuntu : très populaire ...

et bien d'autres... (dont Android des smartphones), il suffit de consulter l'arbre des distributions GNU/Linux sur *http://fr.wikipedia.org/wiki/Distribution_Linux* .

https://linux.developpez.com/actu/137043/Quelles-sont-vos-distributions-Linux-preferees-Et-pour-quelles-utilisations-Merci-de-partager-votre-experience-avec-les-distributions-Linux/

Le but est de développer des environnements utilisateurs et développeurs standard. Une autre approche est de standardiser les systèmes.

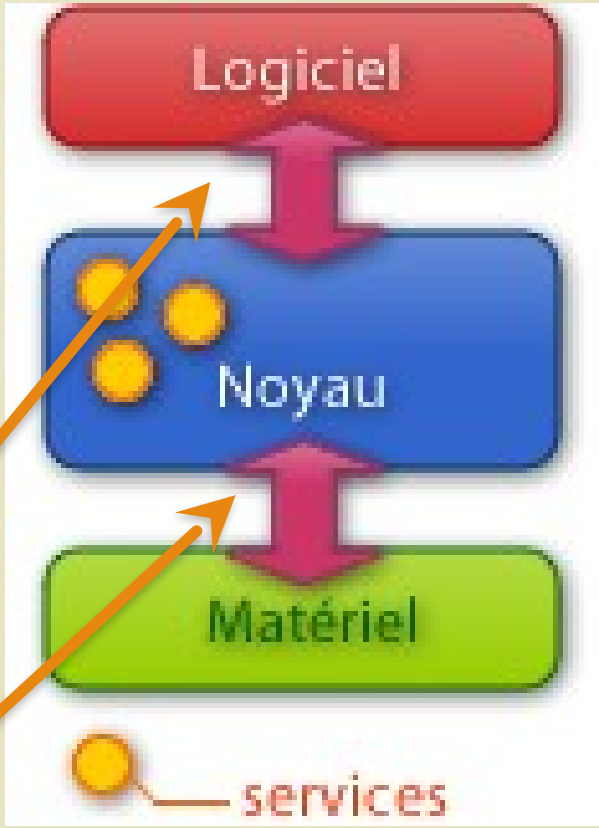
- ***1ère étape*** : Le SVID (System V Interface Definition) d'AT&T en 1985.
- ***2ème étape*** : POSIX (Portable Operating System Interface X) est une interface du système issu d'un groupe d'utilisateurs (/usr/group standard) américains. Cette interface est labélisée par l'ANSI (American National Standard Institute) et l'ISO (International Standard Organisation).
- ***3ème étape*** : X/Open est un consortium de constructeurs qui a pour but de définir un environnement commun de développement (COSE). UNIX devient un label que seul X/Open est habilité à donner.
- ***4ème étape*** : OSF (Open Software Foundation) est créé par IBM et six constructeurs pour développer et proposer un environnement ouvert (logiciels et matériels hétérogènes).
- ***5ème étape*** : en réponse à l'OSF, AT&T et 18 constructeurs et éditeurs, créent ARCHER qui devient UNIX International
- ***6ème étape*** : En juillet 1994 X/Open finalise la liste des interfaces de programmation (API) constituant les *Spec1170*.

- Code source facile à lire et à modifier ; disponible commercialement.
- Interface utilisateur simple ; non-conviviale mais très puissante.
- Le système est construit sur un petit nombre de primitives de base ; de nombreuses combinaisons possibles entre programmes.
- Les fichiers ne sont pas structurés au niveau des données, ce qui favorise une utilisation simple.
- Toutes les interfaces avec les périphériques sont unifiées (système de fichier).
- Le programmeur n'a jamais à se soucier de l'architecture de la machine sur laquelle il travaille.
- C'est un système disponible sur de nombreuses machines, allant du super-calculateur au micro- ordinateur (PC), en passant par les *smart phone*.
- Les utilitaires et programmes proposés en standard sont très nombreux.

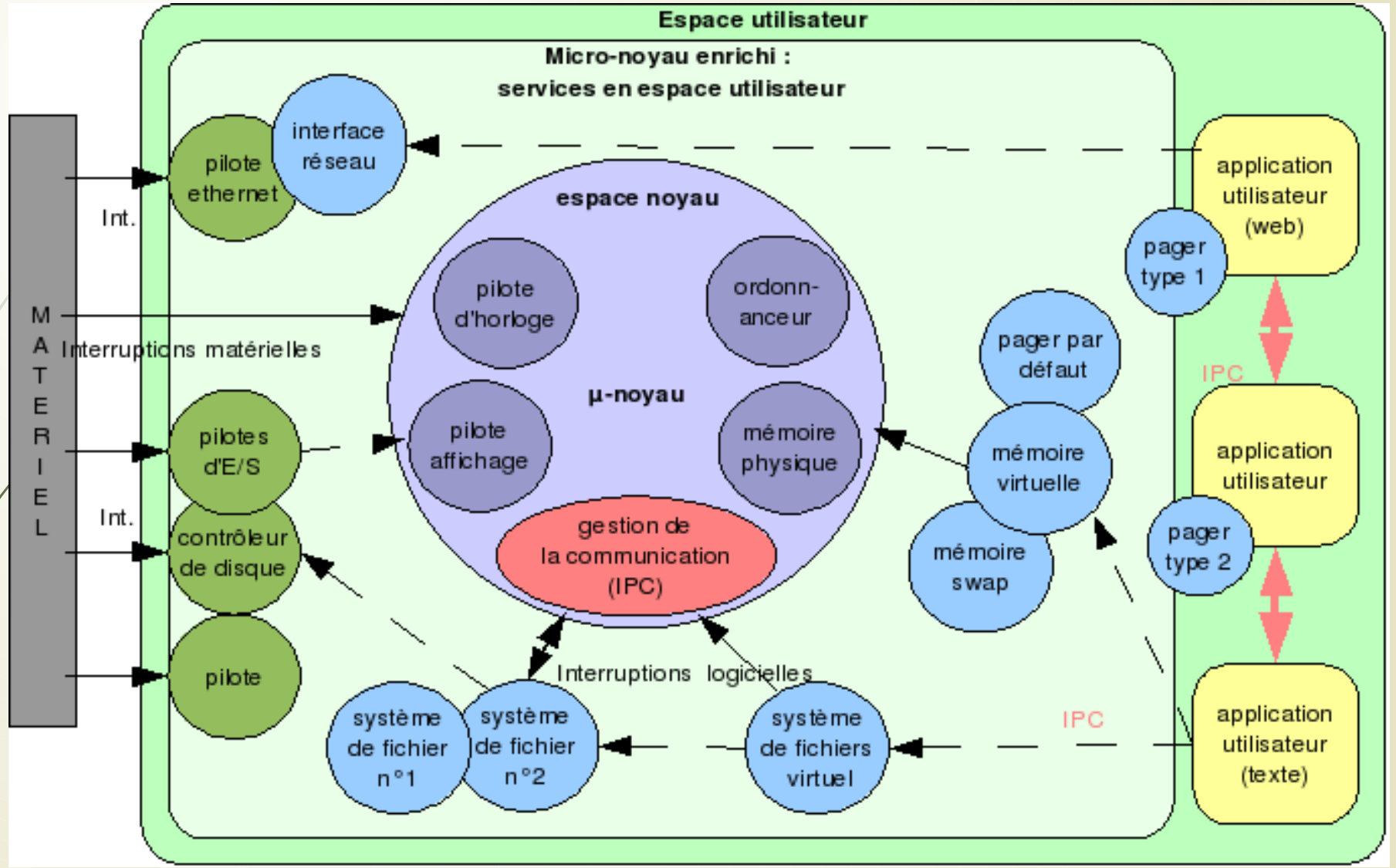
- ***Multi-tâche / multi-utilisateur***
  - Plusieurs utilisateurs peuvent travailler en même temps ; chaque utilisateur peut effectuer une ou plusieurs tâches en même temps.
  - Une tâche ou un processus = programme s'exécutant dans un environnement spécifique.
  - Les tâches sont protégées ; certaines peuvent communiquer, c-à-d échanger ou partager des données, se synchroniser dans leur exécution ou le partage de ressources. Certaines tâches peuvent être «temps réel».

- ***Système de fichiers arborescent***
  - Arborescence unique de fichiers, même avec plusieurs périphériques (disques) de stockage.
  - Le système de fichiers est interfacé au-dessus de l'interface bloc qui utilise les tampons cache du système. Le rôle du cache est de différer les écritures et d'anticiper les lectures
- ***Entrée/Sorties compatible fichiers, périphériques et processus***
  - Les périphériques sont manipulés comme des fichiers ordinaires.
  - Les canaux de communication entre les processus (pipe) s'utilisent avec les mêmes appels systèmes que ceux destinés à la manipulation des fichiers.

- ***Noyau système***
  - UNIX comprend un noyau (kernel) et des utilitaires.
  - Irremplaçable par l'utilisateur, le noyau gère les processus, les ressources (mémoires, périphériques ...) et les fichiers.
  - Tout autre traitement doit être pris en charge par des utilitaires ; c'est le cas de l'interprète de commande (sh, csh, ksh, tcsh ...).
- ***Interfaces du noyau***
  - L'interface entre le noyau UNIX et les programmes utilisateurs est assurée par un ensemble d'appels systèmes.
  - L'interface entre le noyau UNIX et les périphériques est assurée par les gestionnaires de périphériques (devices driver).



https://fr.wikipedia.org/wiki/Noyau_de_syst%C3%A8me_d%27exploitation



https://fr.wikipedia.org/wiki/Noyau_de_syst%C3%A8me_d%27exploitation

- Multi-tâche/multi-utilisateur/multi-plate-forme,
- Gestion dynamique des pilotes de périphériques (module),
- Conformité au standard POSIX,
- Compatibilité avec UNIX System V et BSD (sources),
- Support des bibliothèques UNIX (COFF et ELF),
- Compatibilité binaire avec SCO,
- Support du standard ISO 9660 (CD-ROM),
- Support de plusieurs systèmes de fichiers (dont Windows, Mac,...),
- Interopérabilité avec Windows, Netware,
- Plate-forme de référence pour les standards Internet (apache, ipchains...).

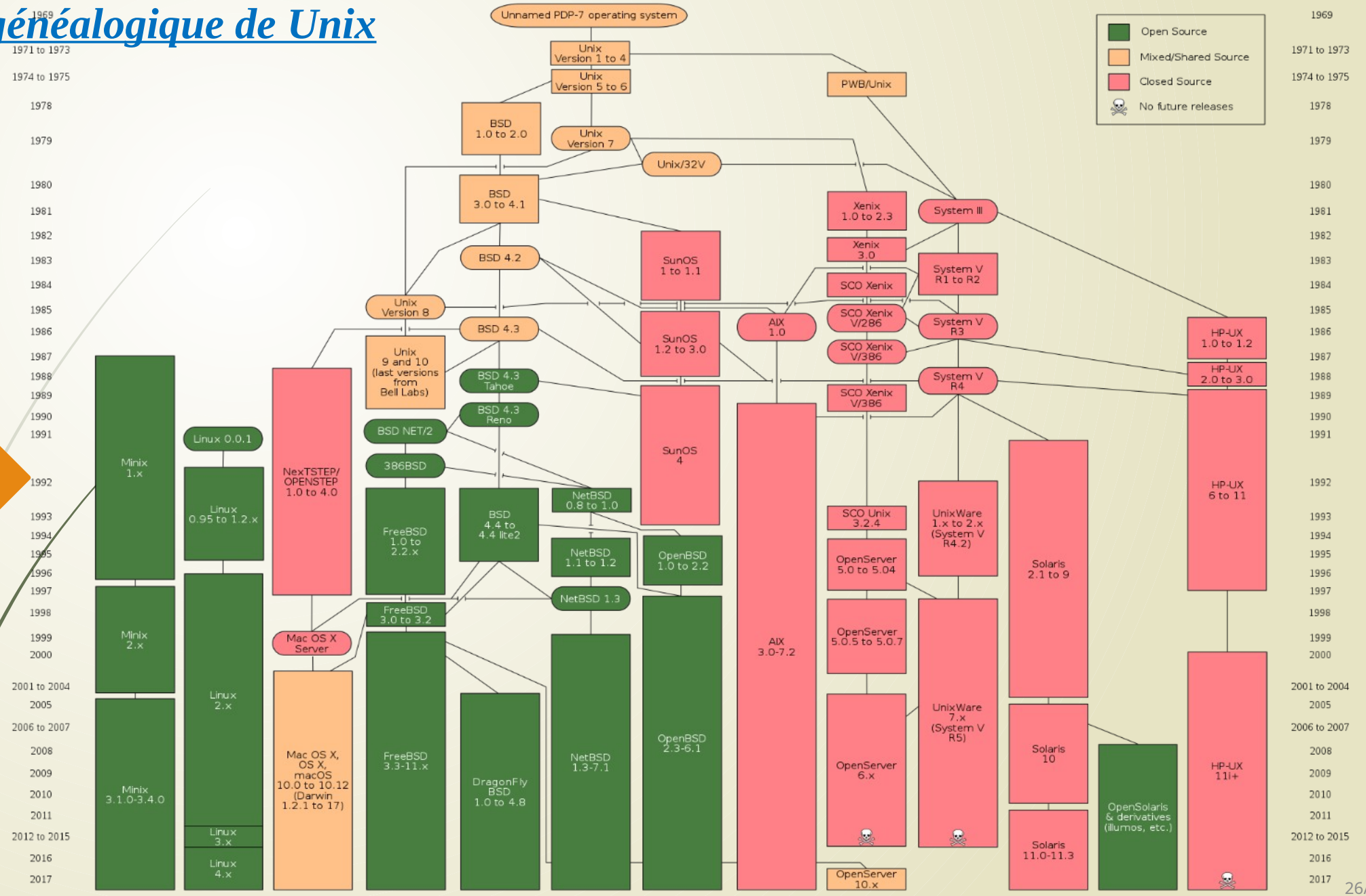
Présentation d'UNIX

Le PDP 7 Oslo

Unix a été développé initialement en 1969 par un groupe d'employés d'AT&T (incluant les créateurs du langage C). Ce système d'exploitation rudimentaire, mais bien fait, tournait sur le PDP-7.

Unix est l'ancêtre de bien des systèmes d'exploitation, notamment Linux. On notera BSD, Solaris, AIX.

Arbre généalogique de Unix



- https://upload.wikimedia.org/wikipedia/commons/d/d9/Unix_history-simple.en.svg

Il contient une interface en ligne de commande.

Unix et le bureau X-Windows vers le début des années 1990.

Linux

La naissance des Linux, un descendant d'Unix, est fascinante : un « hobby » d'un étudiant finlandais (Linus Torvalds) qui cherche à porter le système d'exploitation Unix sur son PC Intel 386. Ce hobby débouche sur un des systèmes d'exploitation les plus importants de l'histoire, première version en 1991. Lisez les débuts ici. Post de Linus Torvalds pour partager son OS,

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID:
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki


Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and
professional like gnu) for 386(486) AT clones. This has been brewing
since april, and is starting to get ready. I'd like any feedback on
things people like/dislike in minix, as my OS resembles it somewhat
(same physical layout of the file-system (due to practical reasons)
among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work.
This implies that I'll get something practical within a few months, and
I'd like to know what features most people would want. Any suggestions
are welcome, but I won't promise I'll implement them :-)
```

Source : l'archive Usenet de Google

GNU

GNU est un projet de création de logiciels libres de droits, souvent associé à Linux.

Une distribution Linux (ou distro), est un ensemble cohérent de logiciels (souvent du GNU) assemblé autour d'un cœur Linux. Il existe plusieurs distributions, dont les principales sont celles-ci. Nous utiliserons surtout Debian et Ubuntu à l'IUT. Si vous le voulez, vous pouvez vous aussi installer Ubuntu à la maison, la distribution à la mode ces temps-ci.

Mac OS X

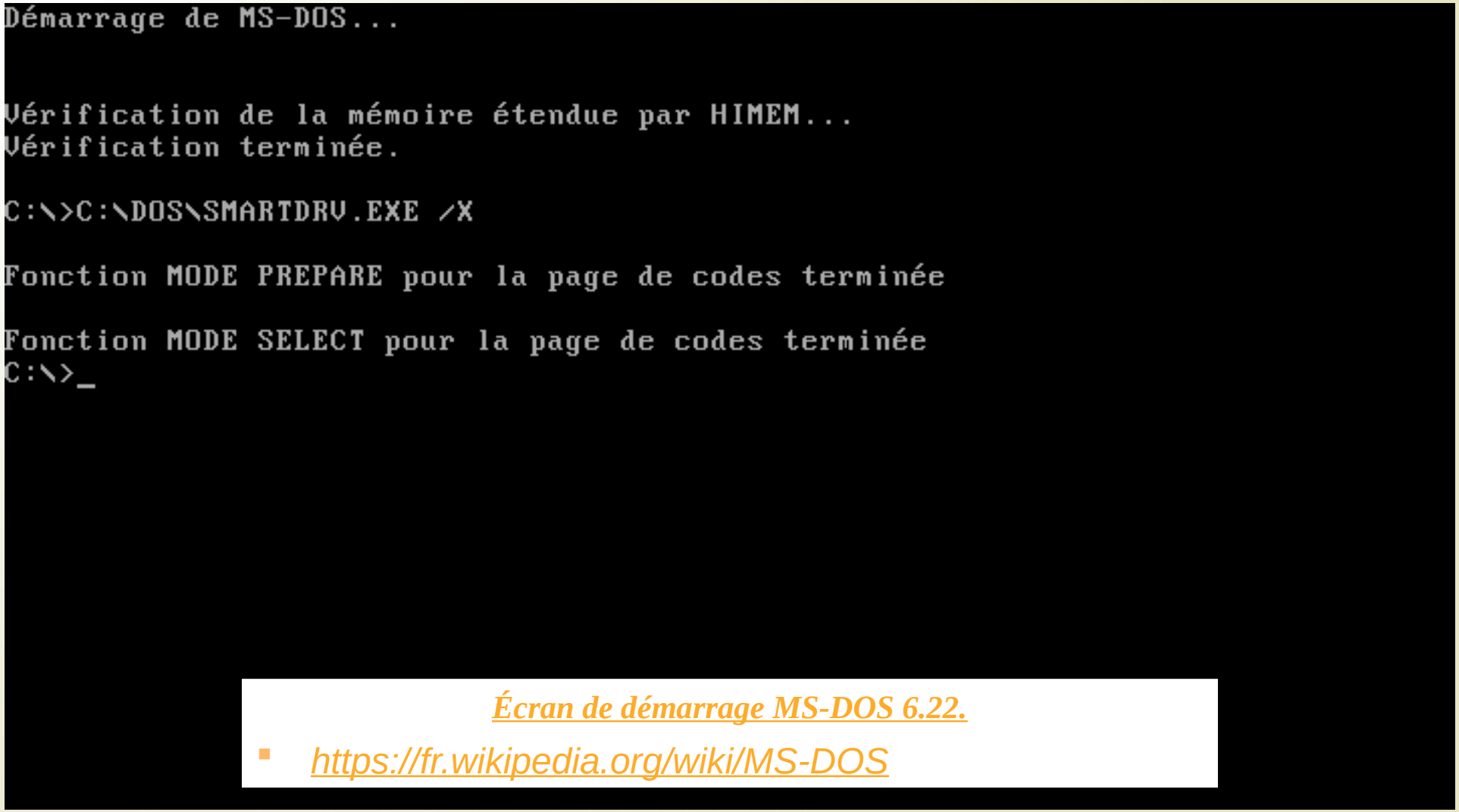
Le système d'exploitation des ordinateurs Macintosh d'Apple est Mac OS X. Certaines parties de FreeBSD (une saveur d'Unix) ont été réutilisées par Apple dans ce système d'exploitation. Une liste complète des versions de Mac OS X et macOS

- https://www.imymac.fr/mac-tips/mac-os-versions.html

Windows : la vache à lait de Microsoft

L'histoire de Windows est tout aussi fascinante, quoiqu'un peu plus mercantile.

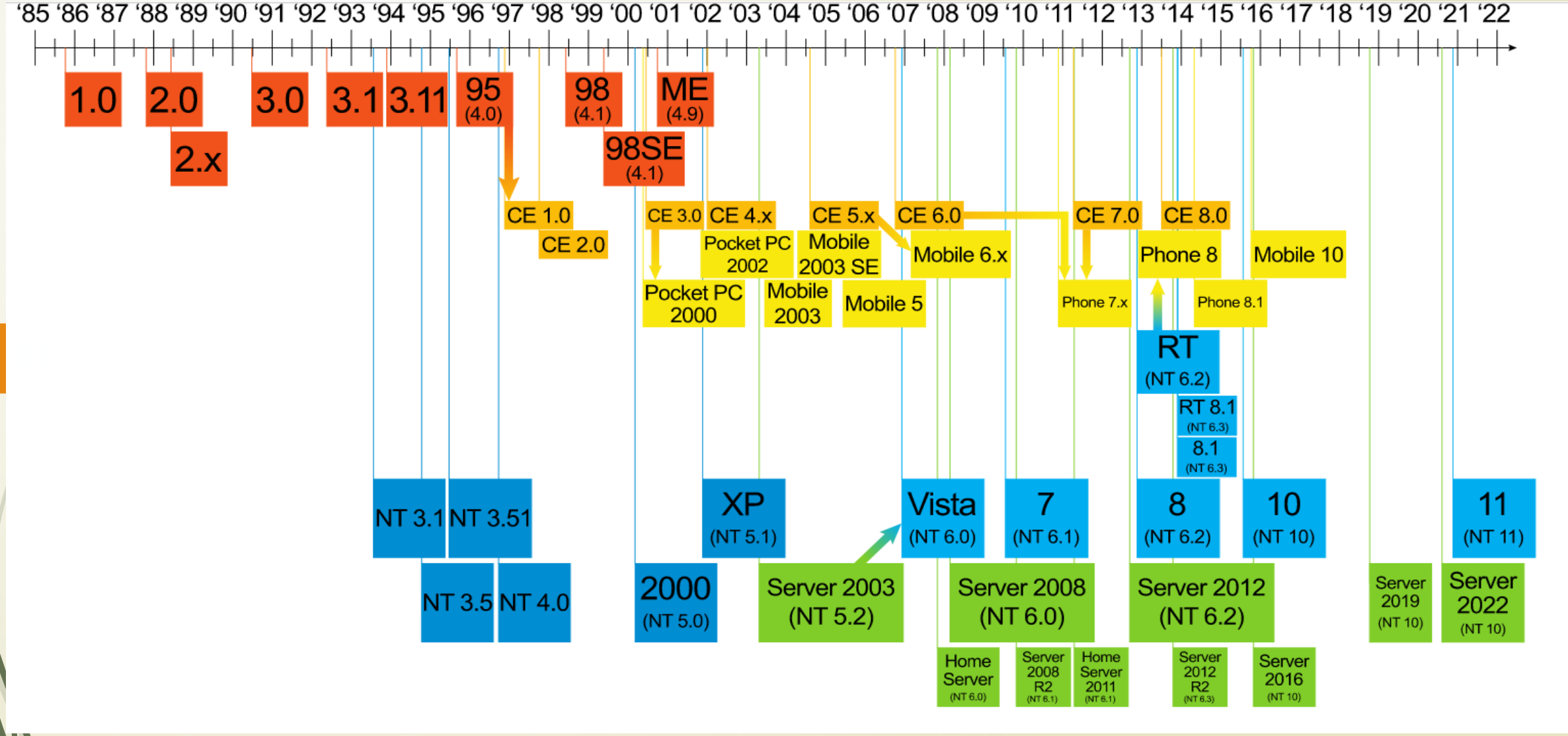
MS-DOS :



Écran de démarrage MS-DOS 6.22.

- *https://fr.wikipedia.org/wiki/MS-DOS*

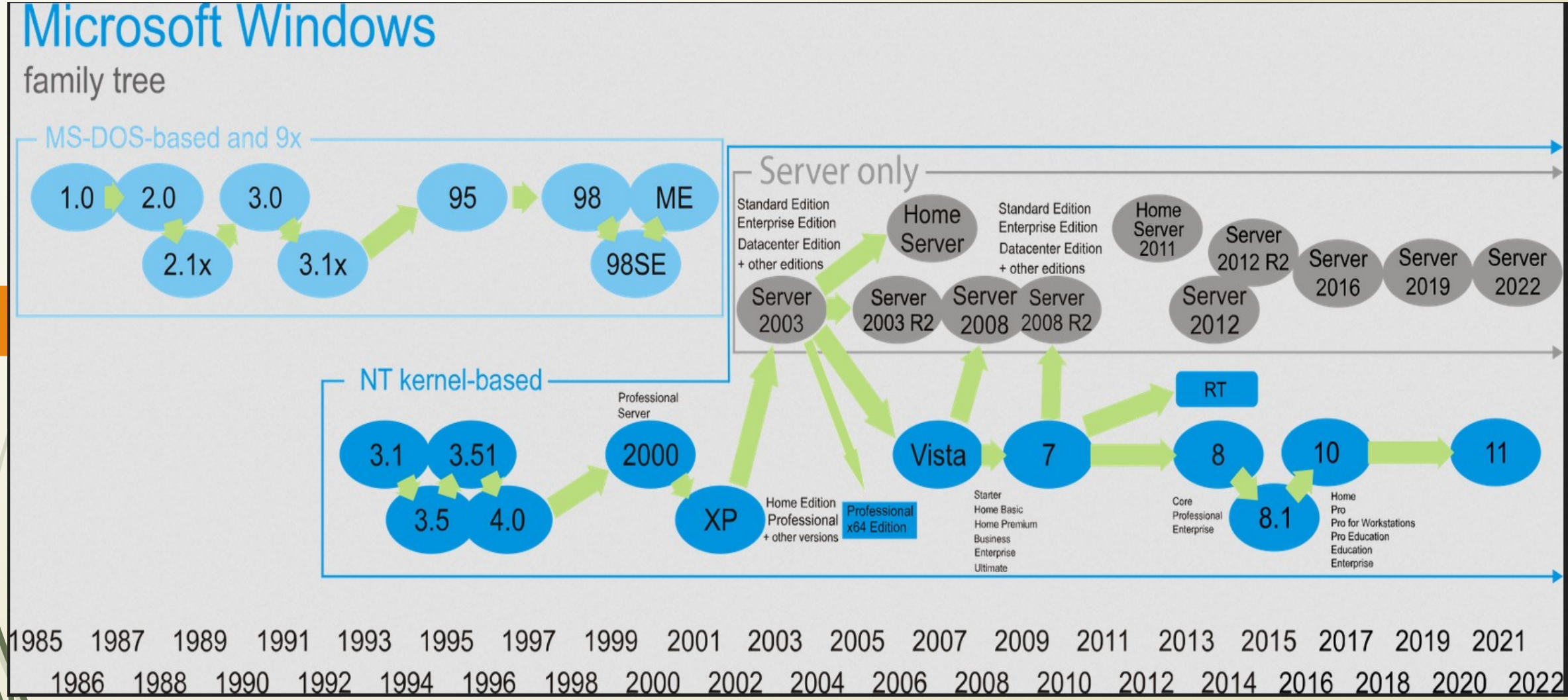
Windows : l'arbre familial : Versions



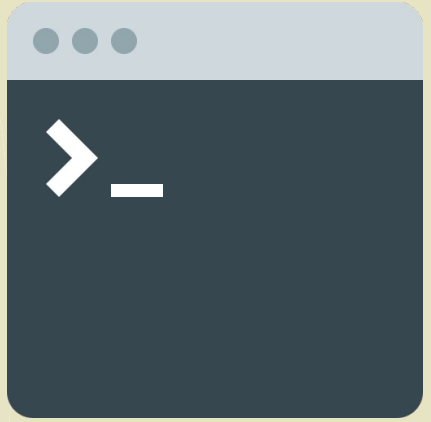
JFA -

https://upload.wikimedia.org/wikipedia/commons/1/17/Suite_des_versions_de_Windows.svg

Windows : l'arbre familial : Noyau



https://upload.wikimedia.org/wikipedia/commons/thumb/0/0e/Windows_Family_Tree.svg/1889px-Windows_Family_Tree.svg.png



Linux
Le ~~petit~~ grand frère de UNIX

JFA - 63

DUT Informatique – Semestre 1

Ressource R1.04

Responsable : Jean-François ANNE

Mise à jour du system Linux

Il faut mettre régulièrement son système linux à jour pour corriger des bugs, ajouter des logiciels, supprimer les logiciels obsolètes, …

Pour cela on a une commande une commande unique : **apt**

En effet **apt** permet d'installer des logiciels sur les distributions à base de Debian depuis les dépôts (repository).

Quelques rappels concernant les packages de distribution.

Un mainteneur s'occupe de maintenir un package qui permet d'installer un logiciel sur la distribution.

Il s'agit d'un fichier **.deb** que le mainteneur créé et maintient.

Ce fichier permet d'exécuter des scripts afin de placer les fichiers, de configuration et de l'application, dans Linux.

En clair donc d'installer les paquets et applications dans Linux.

Les dépôts

Les **dépôts** (respositories en anglais) sont des serveurs mis en ligne qui stockent tous les .deb d'une distribution.

Ainsi, **apt** se connecte à l'un de ses serveurs afin de télécharger le .deb et l'installer. On appelle cela les sources.

Pour ce fait, apt appelle la commande **dpkg**.

Un dépôt peut stocker plus de 20 000 fichiers .deb.

Enfin **apt** peut aussi mettre à jour une distribution Linux.

Cela consiste donc à télécharger tous les **.deb** de la nouvelle version et les installer.

Les logiciels systèmes comme udev, libc ou encore le noyau linux sont aussi mis à jour par cette méthode.

Du côté des distributions Redhat, l'équivalent est **yum**.

Son fonctionnement est assez proche.

Les fichiers de configuration de apt

Afin de fonctionner **apt** utilise plusieurs fichiers et dossiers.

Voici une liste de ces derniers :

- ✓ /etc/apt/sources.list : stocke les sources avec l'adresse des dépôts
- ✓ /etc/apt/sources.list.d/ : sources additionnels. Ainsi on peut ajouter des dépôts non officielles.
- ✓ /etc/apt/apt.conf : fichier de configuration APT
- ✓ /etc/apt/apt.conf.d/ : fichiers de configuration additionnels.
- ✓ /etc/apt/preferences.d/ : Les fichiers de préférences additionnels.
- ✓ /var/cache/apt/archives/ : Stocke les packages (fichiers .deb)
- ✓ /var/lib/apt/lists/ : stocke la liste et informations sur les packages du systèmes.

le dossier **/var/cache/apt/archives** contient les deb déjà téléchargés.

Cela évite en cas de réinstallation un nouveau téléchargement.

Sinon la commande **apt clean** permet de vider ce dossier.

https://www.malekal.com/apt-installer-mise-a-jour-paquet-distribution-debian-ubuntu-mint/

Les commandes apt :

Apt regroupe différentes commandes selon les besoins.

Comme vous modifiez la configuration système, ces opérations nécessite des droits root.

Ainsi sur Debian, il faut s'identifier en root ou utiliser sudo pour les distributions basés sur ce dernier.

- **sudo apt update**

apt update permet de mettre à jour l'**indexation du dépôt** sur votre Linux.

En effet, lorsque votre indexation est trop ancienne, il n'est plus synchronisé avec celui en ligne. Ainsi, vous pouvez demander une version qui n'existe plus et obtenir une erreur.

Enfin on utilise **apt update** lorsque l'on modifie le sources afin de télécharger les nouveaux index.

Cette commande resynchronise votre indexation.

https://www.malekal.com/apt-installer-mise-a-jour-paquet-distribution-debian-ubuntu-mint/

Les commandes apt :

- **sudo apt upgrade**

Cette commande met à jour les paquets de la distribution Linux de votre machine.

Cela ne change pas la version de la distribution mais mets à jour les paquets.

En effet, des mises à jour de sécurité sont publiées chaque jour.

On lance la mise à jour des paquets avec la commande suivante :

sudo apt upgrade

La liste des mises à jour s'affiche.

Ici celle-ci peut être très longue si la dernière mise à jour remonte à très longtemps.

Puis validez par o (ou y)

puis les paquets se téléchargent.

La vitesse et le délai s'affichent en bas à droite de l'écran.

Enfin la phase d'installation s'effectue.

apt peut poser des questions sur des actions à effectuer durant la mise à jour.

https://www.malekal.com/apt-installer-mise-a-jour-paquet-distribution-debian-ubuntu-mint/

Les commandes apt :

- **sudo apt autoremove**
- **sudo apt autoclean**

Ces commandes permettent de supprimer les fichiers qui ne sont anciens ou plus nécessaires.

autoclean : Tout comme clean, autoclean nettoie le référentiel local des paquets récupérés. La différence est qu'il supprime uniquement les paquets qui ne peuvent plus être téléchargés et qui sont inutiles.

autoremove : apt-get supprime les paquets installés dans le but de satisfaire les dépendances d'autres paquets et qui ne sont plus nécessaires.

- **sudo apt dist_upgrade**

Enfin la commande **apt dist-upgrade** permet la mise à niveau de la distribution.

Vous passez donc à la version suivante de votre distribution.

Ensuite on lance **apt upgrade** ou encore **apt-full upgrade** pour finir la mise à jour,

https://www.malekal.com/apt-installer-mise-a-jour-paquet-distribution-debian-ubuntu-mint/

Qu'est-ce qu'un Shell Unix/Linux ?

Un Shell Unix est un interpréteur de commandes destiné aux systèmes d'exploitation Unix et de type Unix qui permet d'accéder aux fonctionnalités internes du système d'exploitation. Il se présente sous la forme d'une interface en ligne de commande accessible depuis la console ou un terminal. L'utilisateur lance des commandes sous forme d'une entrée texte exécutée ensuite par le shell. Dans les différents systèmes d'exploitation Microsoft Windows, le programme analogue est command.com, ou cmd.exe.

Il existe deux environnements très différents sous Unix :

- l'environnement console
- l'environnement graphique

Et il existe plusieurs environnements console : les Shells.

Les Shells Unix/Linux

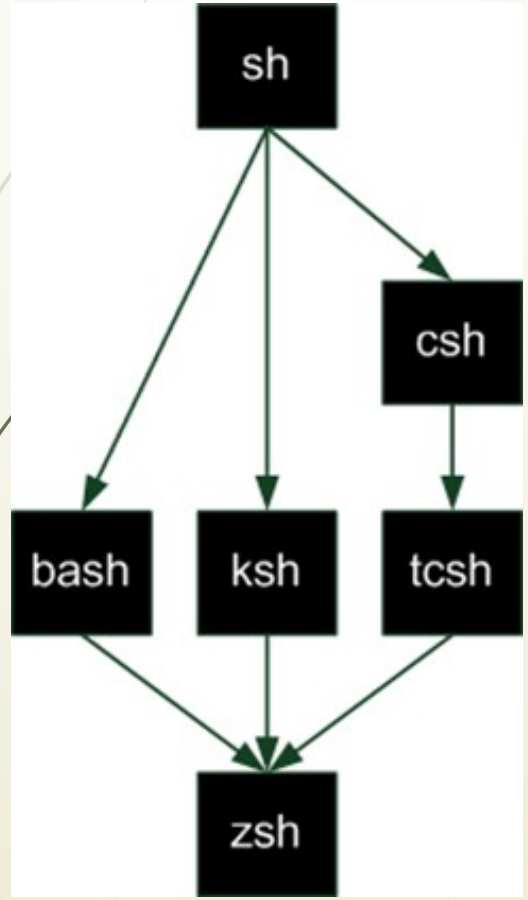
Tous les systèmes d'exploitation de type Unix disposent le plus souvent d'un shell. À l'origine, l'interpréteur de commandes par défaut était sh, qui donna naissance à de nombreuses variantes, dont csh, étendu en tcsh, ou ksh, ou encore rc... Mais aujourd'hui bash, s'inspirant de sh, ksh, et csh, est le shell le plus répandu, bien qu'il existe d'autres interpréteurs de commandes, comme zsh, ou ash.

Voici les noms de quelques-uns des principaux Shells :

- **sh** : Bourne Shell. L'ancêtre de tous les Shells. Sa syntaxe des commandes est proche de celle des premiers UNIX
- **Bash** : Bourne Again Shell. Une amélioration du Bourne Shell augmenté de la plupart des fonctionnalités avancées du C shell, un script Bourne shell sera correctement interprété avec un Bash, disponible par défaut sous GNU/Linux et Mac OS X.
- **ksh** : Korn Shell. Un Shell puissant assez présent sur les Unix propriétaires, mais aussi disponible en version libre, compatible avec Bash.
- **csh** : C Shell. Un Shell utilisant une syntaxe proche du langage C.

Les Shells Unix/Linux

- tcsh : Tenex C Shell. C'est une extension et Amélioration du C shell d'origine.
- zsh : Z Shell. Shell assez récent reprenant les meilleures idées de Bash, ksh et tcsh.



Bash est le Shell par défaut. Cependant, sh reste toujours plus répandu que Bash.

Login

Le premier concept important à garder en mémoire avant de travailler avec Linux est qu'il s'agit d'un système **multiutilisateur**. L'accès à la machine Linux doit donc être **contrôlé** par un compte utilisateur (login, mot de passe) créé par l'administrateur du système.

Chaque usager est connu dans le système par son nom d'utilisateur et son identificateur unique (un nombre entier) qui lui sont attribués par l'administrateur du système.

Il est également affecté à un **groupe** (associé lui-même à un identificateur de groupe unique). Il existe un utilisateur particulier qui dispose de tous les droits : root ou super utilisateur. Le groupe sert généralement à grouper des utilisateurs qui ont un centre d'intérêt commun (projet, facturation) et à en gérer efficacement tous les membres.

La console a toujours un fond noir et un texte blanc(par défaut) . En revanche, les fonctionnalités offertes par l'invite de commandes peuvent varier en fonction du Shell que l'on utilise.

Beaucoup de solutions sont données en ligne de commande, non pas qu'Unix n'ait pas d'interface graphique, mais pour certaines tâches, ***l'utilisation de la ligne de commande s'avère bien plus pratique et plus puissante que la souris***.

Connexion :

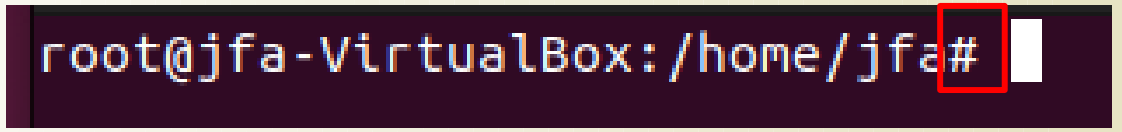
Compte = nom de connexion ou nom d'utilisateur + mot de passe.

Login : Jean

Password : *****

Bienvenue sur …

Prompt>



« $ » (ou « % », « # ») est le prompt ou l'invite de l'interprète de commande utilisé (*shell*) .

L'interprète attend que l'utilisateur tape une commande, exécute cette commande, réaffiche la chaîne d'invite, et attend une nouvelle commande …

^D (CTRL D) en début de ligne => déconnection (fin du shell) ; ce caractère simule une fin de fichier.

^<caractère> correspond à « contrôle caractère ».

Il existe les comptes utilisateurs et le compte super-utilisateur (**root**). L'invite de ce dernier est en général « # » ; ce compte permet d'accéder à tous les fichiers et d'exécuter toutes les manipulations systèmes (administration).

- Modification du mot de passe :

C'est la commande **passwd** :

```
prompt> passwd
Changing password for Jean
Old password : *****
New password : ******
Re-enter new passwd : ******
prompt>
```

- A propos du mot de passe :

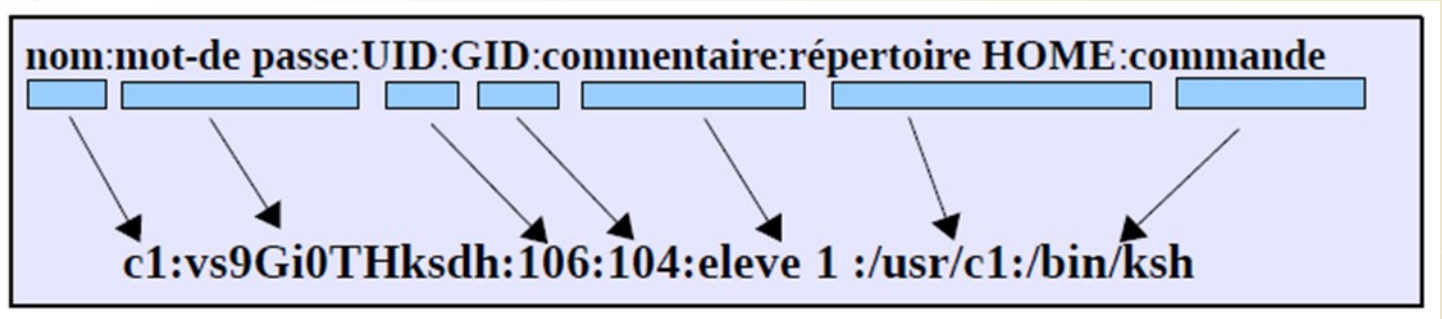
Un bon mot de passe doit être difficile à trouver ! :

Il faut mélanger des chiffres, des lettres, des majuscules/minuscules, mettre des caractères spéciaux, et qu'il ait suffisamment de symboles (plus de 8) !

Sous linux si vous avez oublié votre mot de passe utilisateur, l'administrateur (root) peut vous le changer.

Si c'est le mot de passe root que vous avez oublié, tout est perdu ! Il faut réinstaller le système !

- Ce fichier rassemble des informations sur tous les utilisateurs ayant un compte sur le système.
  - Une ligne par utilisateur et 7 champs par ligne, séparés par le caractère ':'



- Nom : nom de l'utilisateur (connexion) ; 8 caractères au plus (minuscules)
- Mot de passe : soit rempli et crypté (sur 13 caractères) par le programme passwd. Soit contient « x » avec un mot de passe crypté et déporté dans un fichier accessible par l'administrateur
- UID : [User Identification] c'est le numéro d'identification de l'utilisateur (0 pour root).
- GID : [Group Identification] c'est le numéro d'identification du groupe auquel appartient l'utilisateur.
- Commentaire : Champ facultatif.
- Répertoire HOME : Répertoire d'accueil lors de la connexion de l'utilisateur.
- Commande : Commande lancée au moment de la connexion.

Lorsque l'utilisateur se connecte au système sur un terminal texte, plusieurs fichiers sont lus au lancement du shell pour définir l'environnement de travail.

1. /etc/profile

Le fichier /etc/profile est un script shell qui est exécuté en premier lors de la connexion à un terminal texte. Ce fichier contient les variables d'environnement de base de tous les processus, et seul l'administrateur système peut le modifier. En outre, ce fichier exécute des commandes dans l'environnement du shell de connexion.

Ce script n'est interprété qu'à la connexion de l'utilisateur.

2. ~/.bash_profile, ~/.bash_login, ~/.profile

Après lecture du fichier /etc/profile, Bash recherche le fichier ~/.bash_ profile, ~/.bash_login ou ~/.profile dans cet ordre et exécute les commandes contenues dans le premier de ces scripts trouvé et accessible en lecture.

Ce fichier a la même fonction que le fichier /etc/profile, à la différence près qu'il peut être modifié par l'utilisateur pour changer son propre environnement.

Comme le fichier précédent, ce script n'est interprété qu'à la connexion ; les modifications apportées ne sont prises en compte qu'après reconnexion de l'utilisateur.

3. ~/.bashrc

Le fichier ~/.profile n'est exécuté qu'à la connexion. Si l'utilisateur dispose d'un environnement...

L'interface entre UNIX et les terminaux (*tty* abréviation de *teletype*) est prise en charge par un gestionnaire de ligne série (*driver tty*). Celui-ci est paramétrable :

- **erase** annule le dernier caractère frappé (**#**, **^H** ou **DEL**),
- **kill** annule tous les caractères de la ligne (**@** ou **^U**),
- **intr** envoie le signal « **interrupt** » ; interrompt la commande en cours (**DEL** ou **^C**),
- **quit** envoie le signal « **quit** » ; provoque la fin du processus courant (**^|** ou **^**),
- **eof** simule une fin de fichier à partir d'un terminal (**^D**),
- **tabs** indique si le terminal sait faire des tabulations (stty tabs),
- **speed** indique la vitesse du terminal (stty speed).
- Plus de commandes :

  - https://ss64.com/bash/syntax-keyboard.html

Pour voir les paramètres courants, tapez :

```
prompt> stty
speed 9600 baud; ...
```

Pour voir tous les paramètres, tapez :

```
prompt> stty -a
speed 9600 baud; ... line = 0; ...
```


Pour modifier un paramètre, deux cas sont possibles :

- si le paramètre a une valeur :

```
stty nom_du_paramère valeur
```

- si ce paramètre est un booléen :

```
stty nom_du_paramètre, ou
stty -nom_du_paramètre
```

Syntaxe générale d'une commande:

commande [option ...] [argument ...] cd est une primitive du shell

- **Une commande peut être suivie d'options qui précisent la façon dont la commande doit travailler. .**

Exemple :

```
prompt> wc -l fichier
8 fichier
prompt>
```


L'option -l de la commande wc indique que l'on ne doit afficher que le nombre de lignes.

Les options ont le format suivant : -l [arg]

- l est une lettre majuscule ou minuscule.
- arg est facultatif ([ ]). Il permet de passer un argument à l'option

Il peut être séparé ou non de l'option par un blanc. Sauf remarque particulière, les options se situent OBLIGATOIREMENT avant les noms de fichiers.

Cas d'erreurs d'une commande:

Plusieurs cas d'erreurs peuvent se produire :

- La commande n'existe pas.

  ATTENTION le Shell distingue les lettres majuscules des minuscules.

- Vous n'avez pas le droit d'exécuter cette commande.
- Les options de la commande sont erronées.

  En général la commande affiche un message d'utilisation, précisant quels sont les options et arguments. Pour plus de détails taper

  prompt> man commande

- Les arguments de la commande sont erronés

I. Redirection d'entrées/sorties :

Lorsqu'une commande est lancée, ses entrées/sorties peuvent être redirigées sur un fichier.

Exemple 1 :

```
prompt> date > date.out
prompt> cat date.out
Wed Jul 6 11:34:52 MET 1995
prompt>
```

Dans ce cas les données produites par la commande ***date*** ne sont pas affichées sur l'écran, mais écrites dans le fichier ***date.out*** grâce à la redirection **>**.

Exemple 2 :

```
prompt> mail c1 < reponse
prompt>
```

La commande **mail** lit le texte de la lettre à envoyer depuis le fichier **réponse**, grâce à la redirection **<**, au lieu de lire les données à partir du terminal.

Exemple 3 :

```
prompt> wc –l < fichier > nombre
```


La commande **wc** compte le nombre de lignes du fichier « fichier » et envoie le résultat dans le fichier « nombre »

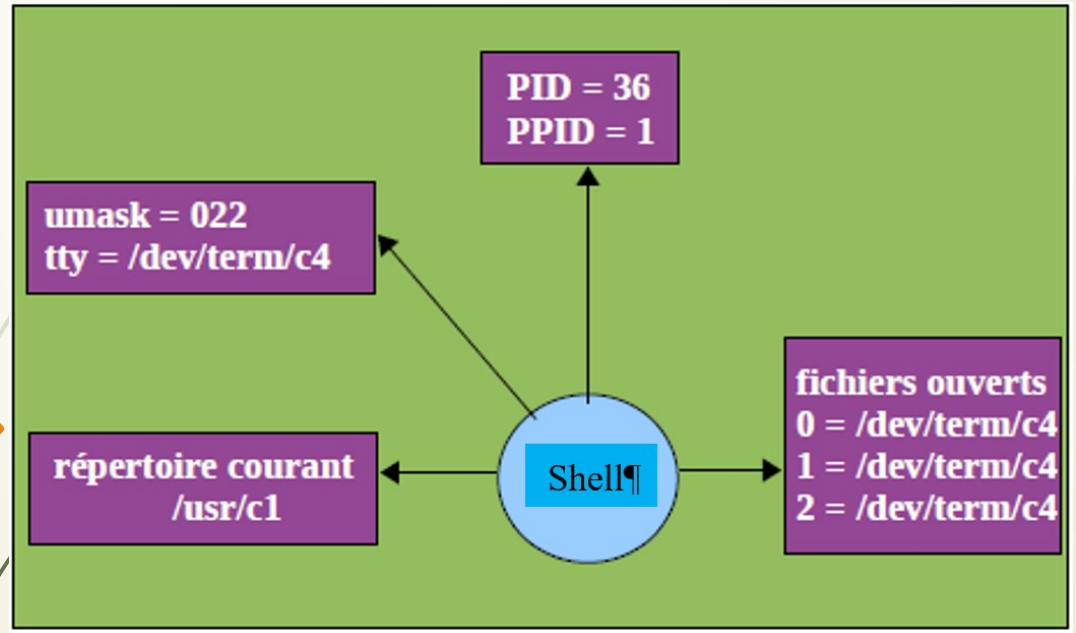
- **Une commande interne est une commande intégrée au shell (built-in command), autrement dit un mot clé ou mot réservé du langage shell. Pour traiter une commande interne, le shell effectue lui-même un traitement spécifique, sans créer de sous-processus.**
- **Une commande interne peut donc modifier l'environnement du processus courant.**
- **Parmi les commandes déjà étudiées, cd, echo, nice, nohup, pwd, test, trap, type, umask, wait sont des commandes internes. Certaines comme cd, trap et umask doivent être internes, d'autres comme echo, pwd et test pourraient être externes, mais ont été intégrées au shell pour des raisons de rapidité d'exécution.**
- Selon le shell utilisé, les commandes internes peuvent être différentes.

Exemple :

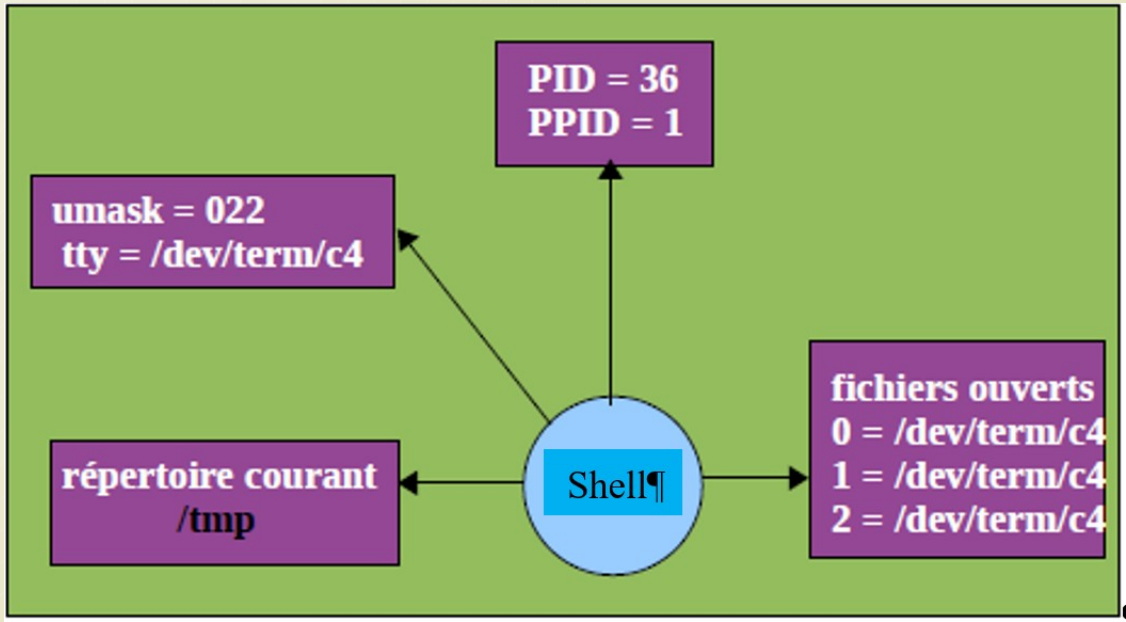
```
prompt> type cd
cd est une primitive du shell
prompt>
```

la commande **type** indique que **cd** est une commande intégrée au shell. Certaines commandes internes possèdent une page au manuel spécifique. Pour les autres vous devez vous reporter aux pages de manuel concernant le shell lui-même.

- **Si on exécute une commande interne, il n'y a pas eu de création de sous- processus :**



- **Avant :** après exécution de la commande cd /tmp, on obtient **:**



- **Après :** Il n'y a pas eu de création de sous-processus, mais le shell a modifié lui-même un des paramètres de son environnement.

- Une commande externe est un fichier exécutable. Le shell recherche ce fichier exécutable dans chacun des répertoires spécifiés par la variable **PATH**. Pour traiter une commande, le shell crée un sous-processus. Cette commande ne peut donc en aucun cas modifier l'environnement du processus. Parmi les commandes :

  who, **date**, **ls**, **cp** ... sont des commandes externes.

- ***Exemple :***

```
jfa@jfa-VirtualBox:~$ type cp
cp est /usr/bin/cp
jfa@jfa-VirtualBox:~$ ls -l /usr/bin/cp
-rwxr-xr-x 1 root root 141824 févr.  7  2022 /usr/bin/cp
jfa@jfa-VirtualBox:~$ file /usr/bin/cp
/usr/bin/cp: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dyn
y linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=ed9fc175
fdfe02c00607c19b011f27efa, for GNU/Linux 3.2.0, stripped
jfa@jfa-VirtualBox:~$
```

- type indique le chemin absolu du fichier exécutable qui constitue la commande.
- ls -l confirme que le fichier existe et vous permet de voir les droits d'exécution.
- file vous confirme qu'il s'agit d'un fichier contenant du code machine exécutable. De plus, vous pouvez consulter le manuel en tapant « man cp ».
- Il existe une autre possibilité pour exécuter une commande externe : **exec**

- Il existe une autre possibilité pour exécuter une commande externe : **exec**
- La commande exec sous Linux est utilisée pour exécuter une commande à partir du bash lui-même. Cette commande ne crée pas de nouveau processus, elle remplace simplement le bash par la commande à exécuter. Si la commande exec réussit, elle ne revient pas au processus appelant.

  Exemple :

  $ ps

  PID TTY TIME COMD

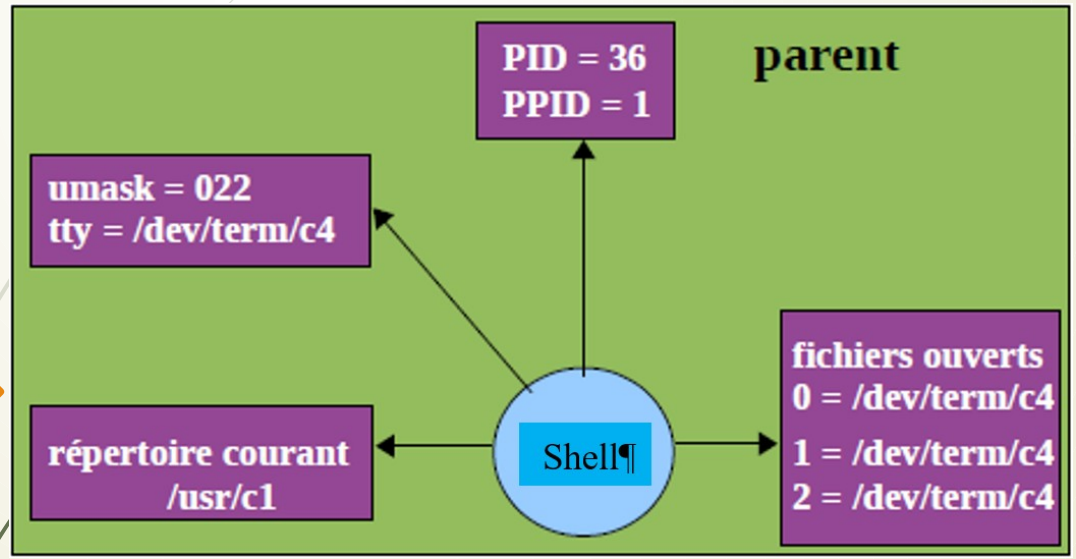
  5244 term/c4 0:00 ps

  122 term/c4 0:05 ksh

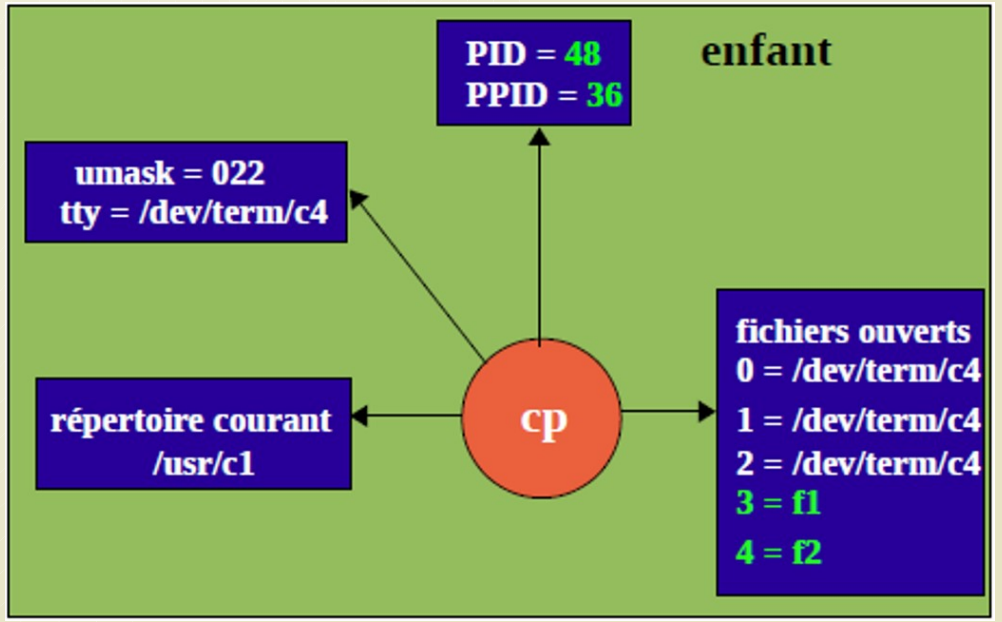
  $ exec cp f1 f2

- Dans ce cas, le processus 122, qui exécutait la commande ksh dans un certain environnement, s'est mis à exécuter à la place du shell, la commande cp. Puis lorsque celle-ci s'est terminée, le processus s'est lui aussi terminé, et donc l'utilisateur s'est trouvé déconnecté !

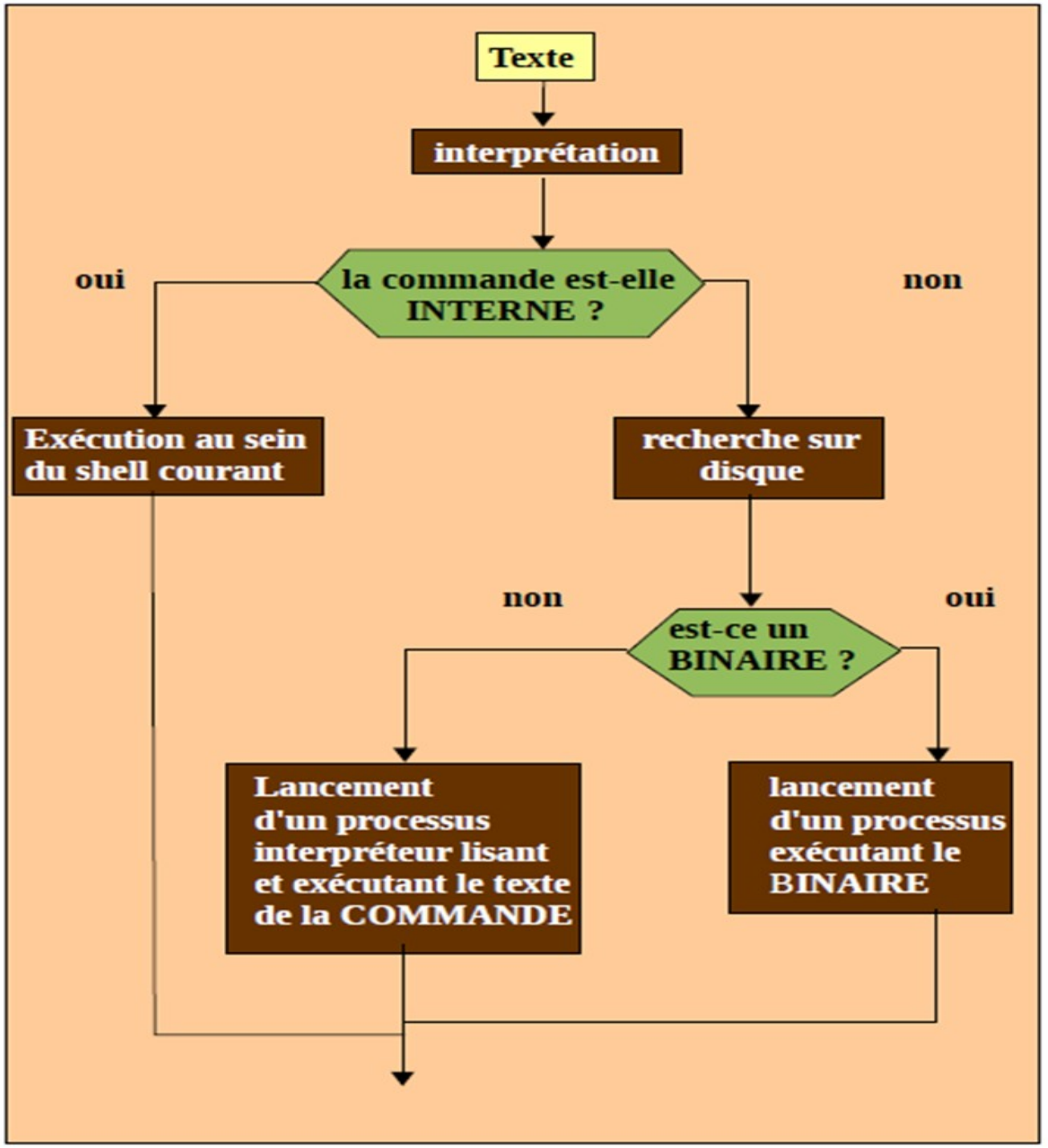
- **Si on exécute une commande interne, il n'y a pas eu de création de sous- processus :**



- **Avant :** pendant l'exécution de la commande "cp f1 f2": **:**



- **Après :** Il y a eu de création d'un sous-processus, et à la fin de la commande, on revient au processus père,


Texte
interprétation
la commande est-elle INTERNE ?
oui
non
Exécution au sein du shell courant
recherche sur disque
non
oui
est-ce un BINAIRE ?
Lancement d'un processus interpréteur lisant et exécutant le texte de la COMMANDE
lancement d'un processus exécutant le BINAIRE

Quelques commandes utiles dès le premier contact avec le système :

- **Uname** : informations relatives à la version du système,
- **Date** : date et heure connues par le système,
- **Tty** : nom de la ligne sur laquelle on est connecté,
- **Who** : liste des utilisateurs connectés,
- **Grep** : recherche les fichiers possédant des chaînes de caractères données,
- **Cal** : calendrier des mois de l'année demandée,
- **Pg** : affiche le contenu du fichier passé en argument par pages de 23 lignes,
- **Man** : pages du manuel correspondant à la commande demandée,
- **Pr** : formatte le texte du fichier passé en argument pour l'impression,

Quelques commandes utiles dès le premier contact avec le système :

- **Uname** : informations relatives à la version du système,

Exemple d'utilisation

```
jfa@jfa-VB:~$ uname -a
Linux jfa-VB 5.11.0-37-generic #41~20.04.2-Ubuntu SMP Fri Sep 24 09:06:38 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
jfa@jfa-VB:~$
```

L'option **-a (all)** affiche toutes les informations.

- **Date** : date et heure connues par le système,

Exemple d'utilisation

```
jfa@jfa-VB:~$ date
mar. 12 oct. 2021 12:37:16 CEST
jfa@jfa-VB:~$
```

MET est le nom du fuseau horaire, ici Mean Europeen Time.

- **Tty** : nom de la ligne sur laquelle on est connecté,

Exemple d'utilisation

```
jfa@jfa-VB:~$ tty
/dev/pts/0
jfa@jfa-VB:~$
```

/dev/ttyp0 est le nom de la ligne série. En effet sous UNIX les périphériques sont désignés par des noms de fichiers.

- **Who** : liste des utilisateurs connectés,

Exemple d'utilisation

```
jfa@jfa-VB:~$ who
jfa      :0           2021-10-12 12:29 (:0)
jfa@jfa-VB:~$
```

- **Grep** : recherche les fichiers possédant des chaînes de caractères données,

Exemple d'utilisation

```
prompt> grep c1 /etc/passwd
c1:vs9Fi0TbjD6xg:208:2001:eleve 1:/usr/c1:/bin/ksh
c10:vs9Fi9bjD6xg:209:2001:eleve
10:/usr/c10:/bin/ksh
prompt>
```

Affiche toutes les lignes du fichier /etc/passwd contenant la chaîne c1.

- **Pg** : affiche le contenu du fichier passé en argument par pages de 23 lignes,
- **Pr** : formatte le texte du fichier passé en argument pour l'impression,

- **Cal** : calendrier des mois de l'année demandée,

Exemple d'utilisation

```
jfa@jfa-VB:~$ cal
    Octobre 2021
di lu ma me je ve sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
jfa@jfa-VB:~$
```

- **Man** : pages du manuel correspondant à la commande demandée,

Exemple d'utilisation

prompt> man date

Date© ...

prompt>

Manuel de référence (man) sur la commande date. Pour avoir plus de détails sur l'utilisation de la commande man, faire « $ man man ».

Syntaxes :

$ man [COMMAND NAME]

$ man [OPTION] [COMMAND NAME]

$ man [SECTION-NUM] [COMMAND NAME]

Le manuel de référence contient plusieurs sections :

1. Les commandes utilisateurs
2. Les appels systèmes
3. La bibliothèque C
4. Les fichiers spéciaux : les périphériques
5. Le format des fichiers
6. Les jeux
7. Divers
8. Les commandes d'administration et de maintenance
9. Le noyau

- Affichage de la page du manuel de la commande *mkfs*, section 8 :

  prompt> man 8 mkfs

- Affichage des pages de toutes les sections du manuel de la commande *passwd* :

  prompt> man -a passwd

- Affichage des pages correspondant à la commande *passwd* dans la section 5 du manuel :

  prompt> man 5 passwd

- Trouver toutes les rubriques contenant un mot clé donné :

  prompt> man -k mot-cle-donne

- Description succincte d'une commande :

  prompt> man -f passwd

https://www.geeksforgeeks.org/man-command-in-linux-with-examples/

Quelques commandes utiles dès le premier contact avec le système :

- **Lp** : gère les requêtes d'impression par file d'attente (spooler),
- **Echo** : affiche ses arguments,
- **Wc** : compte le nombre de lignes, de mots et de caractères contenus dans le fichier passé en argument,
- **Sort** : tri le contenu du fichier passé en argument,
- **Mail** : envoie du courrier à un utilisateur,
- **Write** : pour communiquer avec un autre utilisateur connecté sur le système,
- **Mesg** : interdit ou autorise la réception de messages sur son terminal.
- **Halt** : arrêt du système
- **Shutdown** : arrêt du système

Commandes utiles du terminal :

- **Echo** : affiche ses arguments,

Exemple d'utilisation :

```
prompt> echo installation terminée
installation terminée
prompt> echo 'voulez-vous continuer [y]/n?\c'
voulez-vous continuer [y]/n? $
```

Affiche à l'écran la chaîne de caractères suivant la commande « echo », après d'éventuelles interprétations (variables, fonctions ...).

- **wc** : compte le nombre de lignes, de mots et de caractères contenus dans le fichier passé en argument,

Exemple d'utilisation :

```
prompt> wc fichier
8 48 208 fichier
prompt>
```

Affiche les caractéristiques du fichier de nom « fichier » en termes de contenu, de la façon suivante : il possède 8 lignes, 48 mots et 208 caractères.

- **Mail** : envoie du courrier à un utilisateur,

Exemple d'utilisation :

```
prompt> mail c2
Subject: ...
^D
prompt>
```

Pour une communication asynchrone (mode messagerie) ; le destinataire n'est pas nécessairement connecté.

- **Shutdown** : arrêt du système,

Exemple d'utilisation :

```
prompt> prompt> shutdown -h now
```

Pour une communication asynchrone (mode messagerie) ; le destinataire n'est pas nécessairement connecté.

Les personnes « c1 » et « c2 » peuvent communiquer simplement par la fonction « write » :

« c1 » tape :

prompt> write c2

« c2 » reçoit :

« beep » Message from c1 on ttyd1

« c1 » tape alors :

prompt> write c2

A partir de ce moment, tout ce que « c1 » frappe est envoyé à « c2 » dès le retour chariot, et réciproquement. Chacun tape ^D en fin de dialogue. Il s'agit de communications interactives.

```
prompt> mesg is y
prompt>
```

Permet de voir si l'on autorise (y) les messages à être envoyés sur le terminal courant.
Pour les interdire faire :

```
prompt> mesg n
prompt>
```


On peut taper plusieurs commandes sur la même ligne en utilisant le séparateur « ; » :

```
prompt> date; who; tty
```

- **Sum:** Calcule et affiche une somme de contrôle pour un fichier (intégrité des fichiers).

Exemple d'utilisation :

```
prompt> sum fichier
08860    1
prompt>
```


- **Diff** : Affiche les lignes différentes devant être modifiées pour que les deux fichiers soient identiques.

Exemple d'utilisation :

```
prompt> diff fichier fichier 1
6c6
< message
---
message 2
prompt>
```

- **Touch :** Modifie la date de dernière modification du fichier, celle-ci devient égale à la date à laquelle la commande a été exécutée. Si le fichier n'existe pas, il sera créé (et de taille nulle) sauf si l'option c est utilisée.

Exemple d'utilisation :

```
prompt> touch fichier
prompt> ls –l fichier
-rw-r--r-- 1 c1 cours 0 Oct 9 1991 fichier
prompt>


prompt> touch test_{a..j}.txt
prompt> ls –l test*.*
test_a.txt  test_c.txt  test_e.txt  test_g.txt  test_i.txt
test_b.txt  test_d.txt  test_f.txt  test_h.txt  test_j.txt
prompt>
```

- **Cmp** : Compare octet par octet les deux fichiers passés en paramètre. Cette commande renvoie « 0 » si les fichiers sont identiques, « 1 » sinon.

```
cmp fichier1 fichier2
```

Exemple d'utilisation :

```
prompt> cmp file1.txt file2.txt
file1.txt file2.txt differ: byte 9, line 2
 /*indique que la première différence trouvée entre les deux fichiers est le 9ème byte de la ligne 2*/
prompt> cmp file2.txt file3.txt
prompt>
 /*indique que les fichiers sont identiques*/
```

- **Compress** : Permet une opération de compression visant à diminuer l'espace occupé par les différents fichiers -référencés. Chaque fichier est remplacé par un nouveau fichier dont la référence est obtenue en suffixant la référence d'origine avec l'extension .Z. Les caractéristiques du fichier sont conservées. Le fichier d'origine est supprimé.

```
compress [options] liste_fichiers
```

Exemple d'utilisation :

```
prompt> compress -v exemple.xls
exemple.xls :  -- replaced with exemple.xls.Z Compression: 24.57%
prompt>
 /*le fichier exemple,xls est compressé et remplacé par le fichier exemple.xls.Z*/
Prompt> compress -rv abc
prompt>
 /*compresse tous les fichier contenus dans abc et ses sous répertoires de manière récursive*/
```


https://www.geeksforgeeks.org/compress-command-in-linux-with-examples/

- **uncompress** : Permet la décompression et la reconstruction d'une série de fichiers à partir de leurs formes compressées avec la commande compress. .

```
uncompress [options] liste_fichiers
```

Exemple d'utilisation :

```
prompt> uncompress -v exemple.xls.Z
exemple.xls :  24.6% -- replaced with exemple.xls
prompt>
```

/*le fichier exemple.xls.Z est décompressé et remplacé par le fichier exemple.xls*/

https://www.computerhope.com/unix/uuncompr.htm

- **zcat** : Affiche sur la sortie standard le fichier d'origine, correspondant à un fichier compressé au moyen de la commande compress.

```
zact [options] liste_fichiers.Z
```

Exemple d'utilisation :

```
prompt> zcat exemple.Z
Ceci est un fichier exemple
prompt>
/*Permet d'afficher de manière lisible le contenu d'un fichier compressé*/
```


https://www.computerhope.com/unix/uzcat.htm

- **tar :** tar (tape archiver) permet de gérer des archives de fichiers (sur bande, disque ou fichier). La clé définit les actions de la commande. Elle est constituée d'une suite de caractères définissant la fonction (crtux) et des qualificatifs de cette commande (Abfhmov). .

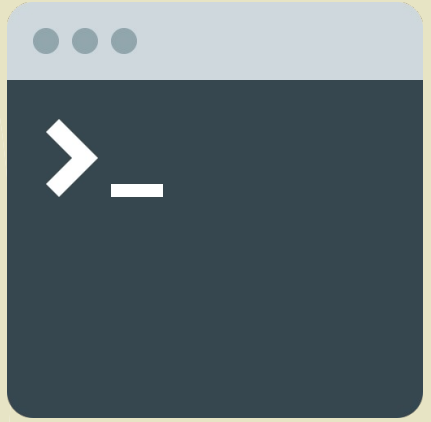
```
tar [options] liste_fichiers.tar
```

Exemple d'utilisation :

```
prompt> tar cvf archive_cible.tar /etc
/* Place tous les fichiers du répertoire /etc dans le fichier "archive_cible.tar". */
prompt> tar xvf archive_cible.tar
/* Extraction de "archive_cible.tar" dans le répertoire courant. */
prompt> tar tvf archive_cible.tar
/* Liste les fichiers contenus dans "archive_cible.tar". */
```

https://www.computerhope.com/unix/uzcat.htm

- **Les différentes fonctions de la commande sont :**
  - **-c : création d'une nouvelle archive. Sur bande, l'écriture de l'archive a lieu en début de bande et non à la suite du dernier fichier ;**
  - **-r : fonction de remplacement permettant d'écrire en fin d'archive les fichiers de références données ;**
  - **-t : liste des références de fichiers dans l'archive sans restitution ;**
  - **-u : les fichiers sont ajoutés en fin d'archive s'ils n'y figurent pas encore ou si la date de modification de la dernière version archivée est antérieure à la version du fichier sur le disque ;**
  - **-x : fonction d'extraction de l'archive. Si la référence examinée est une référence de répertoire, son contenu est extrait de manière récursive. Si aucune référence de fichier n'est donnée, tous les fichiers de l'archive sont extraits.**
- **Un certain nombre d'indications supplémentaires permettent de qualifier la fonction réalisée :**
  - **-A : les messages d'avertissement sont supprimés ;**
  - **-f : l'argument suivant est interprété comme une référence de fichier correspondant au nom de l'archive (au lieu d'une référence par défaut qui est en général celle d'un fichier spécial associé à un dérouleur de bande). Si cet argument est -, la commande lit sur l'entrée standard ou écrit sur la sortie standard ;**
  - **-h : les liens symboliques sont suivis (par défaut, ils ne le sont pas) ;**
  - **-v : option «verbeuse».**



Linux
Le système de fichiers

JFA - 110

DUT Informatique – Semestre 1

Ressource R1.04

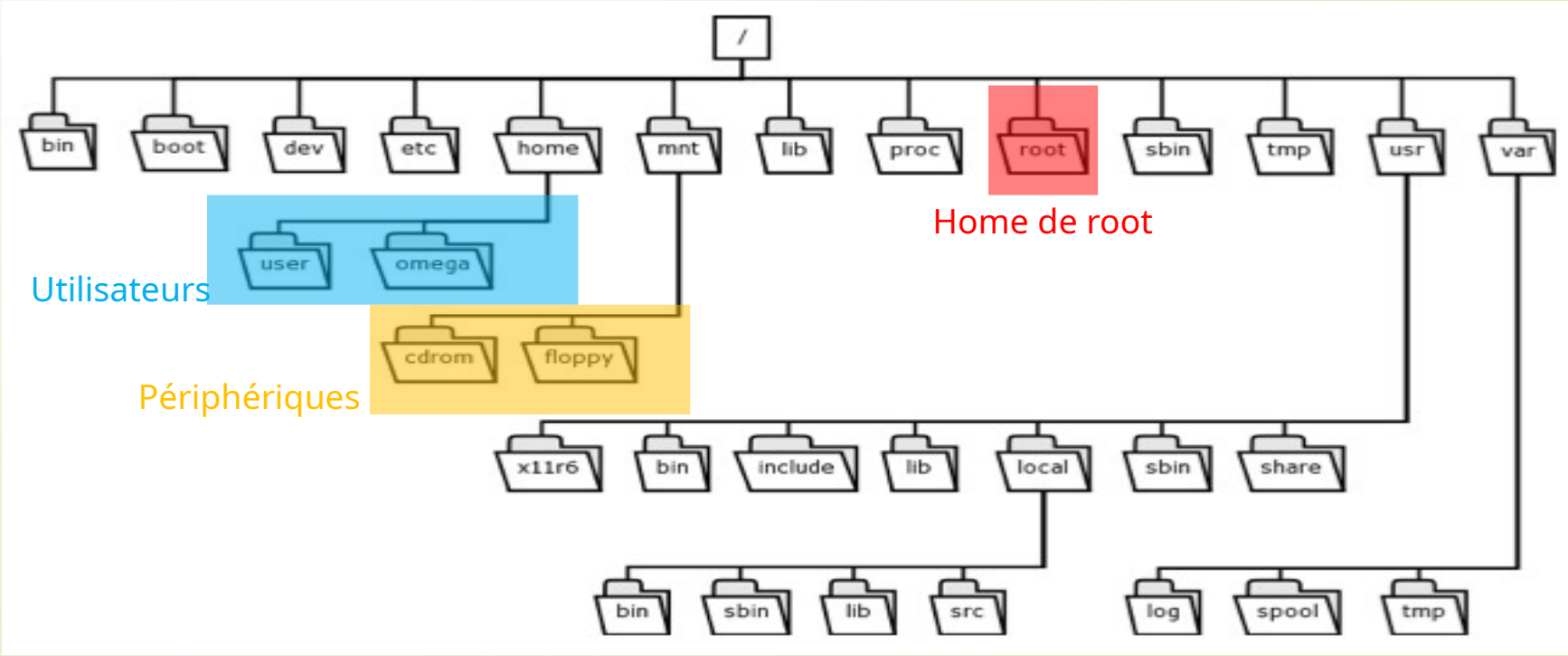
Responsable : Jean-François ANNE

Le système de fichiers, ses commandes associées:

Un système multiutilisateur se doit de ranger ses fichiers correctement pour éviter un désastre.

Sous LINUX, ils sont organisés en une structure hiérarchique appelée arborescence, et rangés dans des répertoires.

Les différents répertoires qui contiennent les fichiers du système et vos propres fichiers, sont organisés en une structure arborescente qui part de la racine root (racine en français) et se déploie sous forme de ramifications multiples. Un répertoire particulier home (maison en français) est réservé par le système pour recevoir les fichiers des différents utilisateurs.



L'organisation des fichiers arborescente :

- **/etc** : les fichiers de configuration,
  - **/etc/rc.d** : le répertoire des scripts associés aux différents niveaux de fonctionnement,
  - **/etc/rc.d/rc3.d** : scripts exécutés au démarrage du niveau 3 de fonctionnement du système,
- **/mnt** : répertoire pour le montage de CD-ROM, disques, partages, …,
- **/proc** : répertoire pour la mémorisation des processus,
- **/root** : le répertoire de connexion de l'administrateur,
- **/bin** : les binaires nécessaires en mode mono-utilisateur,
- **/sbin** : les commandes de démarrage et d'arrêt du système,
- **/tmp** : le répertoire temporaire pour la création de fichiers temporaires,
- **/dev** : le répertoire des fichiers spéciaux,
- **/var** : le répertoire des fichiers des services,
  - **/var/mail** : les boîtes aux lettres des utilisateurs,
  - **/var/spool/cron/** : le répertoire des fichiers de données du service *cron* pour la commande *crontab*,

L'organisation des fichiers arborescente est une organisation récente :

- **/var/spool/lpd** : fichiers de données du service impression,
  - **/var/run/** : fichiers contenant les PID des services actifs et des utilisateurs connectés,
  - **/var/log** : les fichiers de « *log* »,
  - **/var/crash** : les fichiers image mémoire en cas de crash du système,
- **/usr/** : répertoire des commandes du système,
  - **/usr/bin** : commandes de base du système,
  - **/usr/lib** : les bibliothèques de sous-programmes,
  - **/usr/man** : les manuels de référence,
  - **/usr/src** : les fichiers source du noyau,
  - **/usr/src/linux** : répertoire pour recompiler le noyau,
  - **/usr/doc** : la documentation des paquetages,
- **/home :** répertoire des répertoires de connexion des utilisateurs.

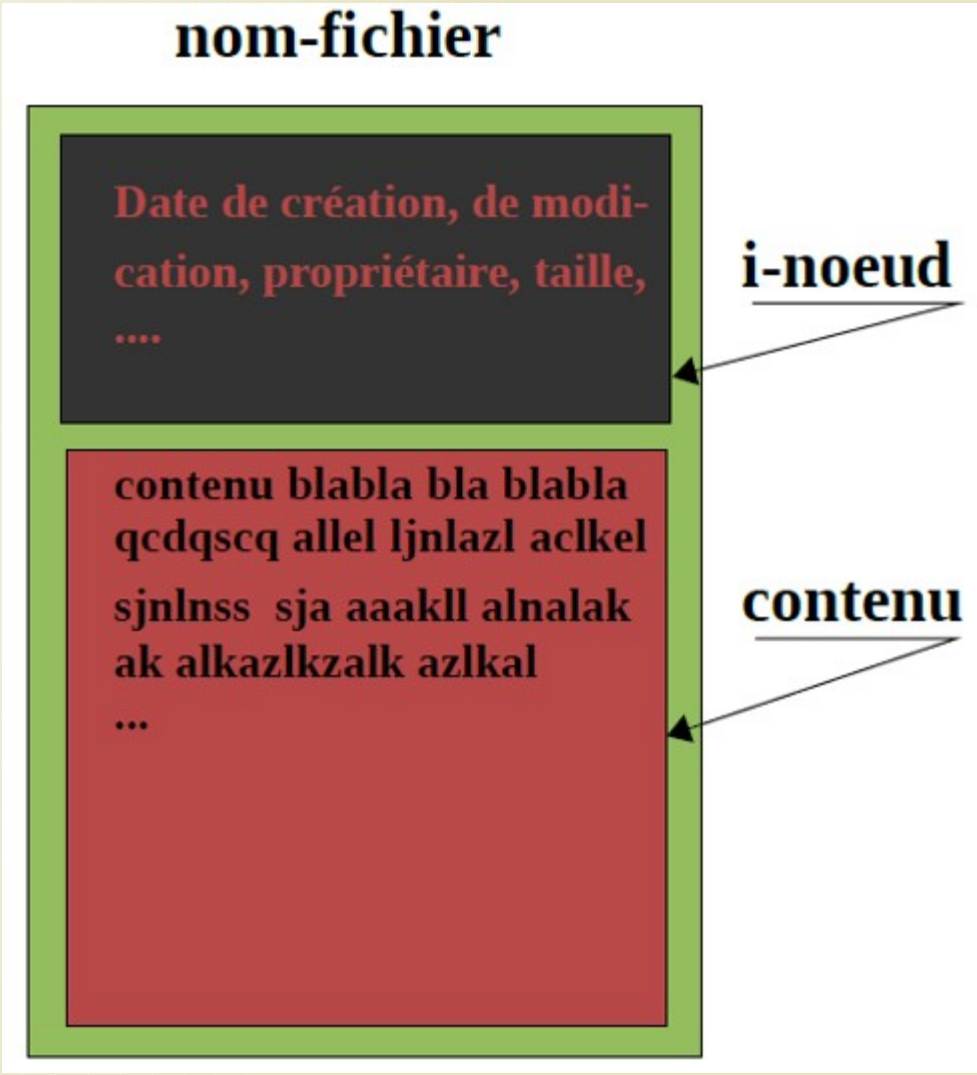
Les fichiers ordinaires :

*Dans Linux et Unix, **tout est fichier**.*

- *Les répertoires sont des fichiers,*
- *les fichiers sont des fichiers,*
- *les périphériques sont des fichiers. Même si les périphériques sont souvent appelés nœuds, ce sont quand même des fichiers.*

Pour le système, les données d'un fichier forment une suite non structurée d'octets (byte stream). Un fichier texte, par exemple, est une suite de codes ASCII, les lignes étant séparées par le code ASCII 'nouvelle ligne' (012 octal). Il n'existe aucune notion d'organisation telle que séquentiel indexé, etc...

Un certain nombre de caractéristiques sont associées à un fichier : la date de sa création, celle de la dernière modification, le propriétaire, la taille, etc... Ces caractéristiques sont regroupées dans un descripteur de fichier, appelé nœud d'index (i-node ou index-node).

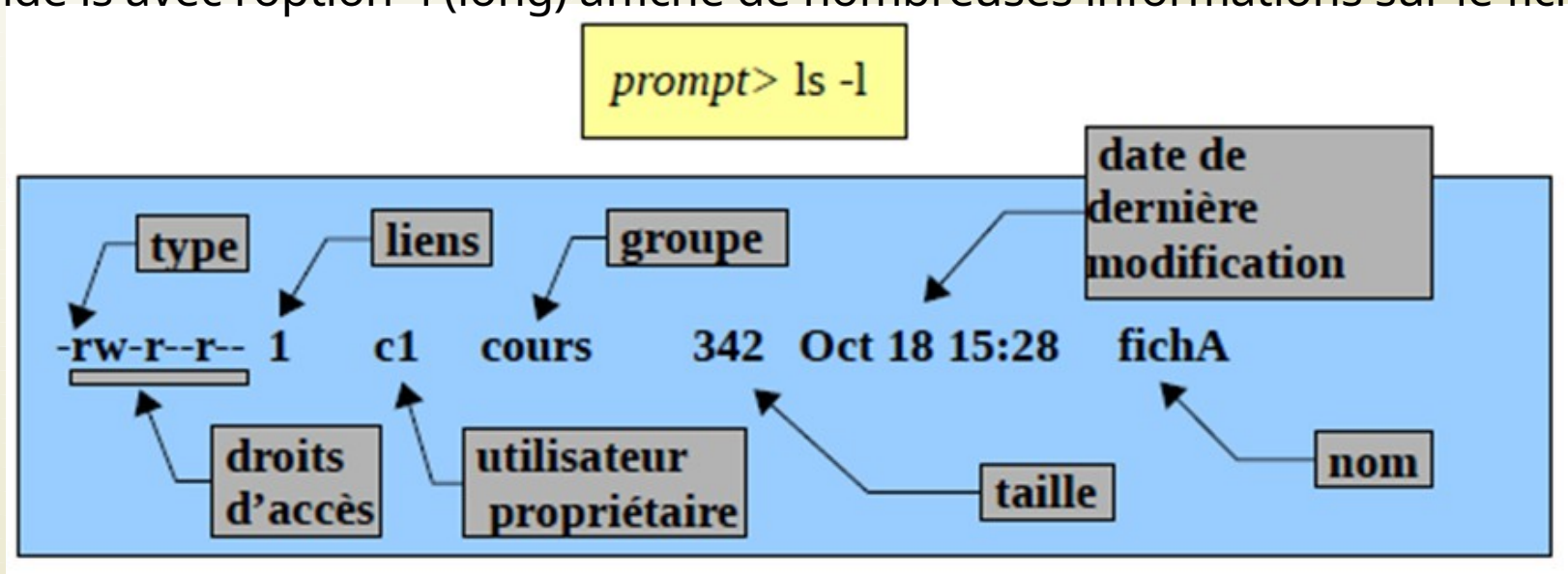


Les fichiers ordinaires : commandes associées

- **ls** *fichier* ... [**ls**t] affiche le contenu des répertoires (à un niveau) et les noms des fichiers passés en argument, c'est_à-dire *fichier* ..., ou s'il n'y a pas d'argument, tous les fichiers du répertoire courant sauf ceux commençant par un point.
- **cat** *fichier* ... [con**cat**enate] affiche le contenu des fichiers *fichier* ...
- **cp** *fichier1 fichier2* [**c**o**p**y] copie *fichier1* dans *fichier2*
- **mv** *fichier1 fichier2* [**m**o**v**e] renomme *fichier1* en *fichier2*
- **rm** *fichier* [re**m**ove] détruit le fichier *fichier* ; irréversible.

Note : **Toutes les commandes UNIX qui manipulent les fichiers acceptent toutes les formes (absolu, relatif ...) de chemins de noms de fichier.**

La commande ls avec l'option -l (long) affiche de nombreuses informations sur le fichier :



Les fichiers ordinaires : commandes associées

- ***Le type du fichier :***
  - 'd' : pour répertoire,
  - '-'pour fichier ordinaire,
  - 'b' pour périphérique bloc,
  - 'c' pour périphérique caractère,
  - 'l' lien symbolique,
  - 'p' tube nommé (IPC),
  - 's' socket locale (IPC).
- ***Le nom de fichier*** : Limité à 14 (ou 255) caractères parmi le jeu ASCII. Le système n'impose aucun format. On évite les caractères invisibles et les méta-caractères (*, ?, [et]).
- ***La taille du fichier*** : C'est son nombre d'octets. Elle sert à déterminer la fin du fichier. Il n'y a donc pas de marque de fin de fichier.
- ***Droits d'accès :*** Trois groupes d'autorisation, l'utilisateur propriétaire, les personnes appartenant au groupe propriétaire et les autres.

Les fichiers ordinaires : droits d'accès

Pour chaque groupe 3 caractères indiquent les autorisations :

- ***r***, ***w*** et ***x***.

- **R** ***lecture*** *:* ***Autorisation de lire le contenu*** (cat, pg ...) ; ce droit est nécessaire pour faire une copie du fichier. ***ls*** permet de lister le contenu du fichier (attention pour *ls -l* il faut aussi le droit *x*).
- **w** ***écriture*** *:* ***Autorisation de modifier le contenu*** ; On peut écrire dans un fichier avec un éditeur ou une commande (cat). Il faut ce droit pour le fichier destination d'une copie, d'un déplacement ou d'un lien, si le fichier existe déjà. Pour un répertoire on peut créer et détruire des fichiers et des répertoires. Lors d'une destruction, seule cette permission prévaut. Elle permettra de détruire un fichier, même si on n'a aucun droit dessus. La destruction d'un fichier revient à modifier le répertoire, pas le fichier !
- **x** ***exécution/traverse*** ***(répertoire)***
- **-** à la place de ***r***, ***w*** ou ***x*** signifie ***non-autorisé***.

Commandes associées

- **ls -F :** affiche les noms de fichiers suivis du caractère '/' s'il s'agit d'un répertoire, et du caractère '*' s'il s'agit d'un fichier ayant la permission d'exécution.
- **ls -C :** affiche les noms de fichiers par colonnes.
- **ls -l :** affiche les noms de fichiers par ligne avec toutes les informations.

Changer les droits d’un fichier ou d’un répertoire

- **chmod : Modifie les droits d'accès sur un fichier ou un répertoire**

chmod code fichier

chmod code repertoire

Exemple d'utilisation :

prompt> chmod 777 fichier

Personne concernée	
propriétaire	u
groupe	g
autres	o
tous	a
Action	
ajouter	+
enlever	-
initialiser	=
Accès autorisés en	
lecture	r
écriture	w
exécution/traverse	x

Changer les droits d'un fichier ou d'un répertoire

Exemple :

```
prompt> ls -l fichA
-rw-rw-rw- 1 c1 cours 342 Oct 18 15:28 fichA
 P  G  A
prompt> chmod go-w fichA; ls -l fichA
-rw-r--r-- 1 c1 cours 342 Oct 18 15:28 fichA
prompt>
prompt> chmod u+x fichA; ls -l fichA
-rwxr--r-- 1 c1 cours 342 Oct 18 15:28 fichA
prompt>
prompt> chmod g-r fichA; ls -l fichA
-rwx---r-- 1 c1 cours 342 Oct 18 15:28 fichA
prompt>
prompt> chmod ug+rw fichA; ls -l fichA
-rwxrw-r-- 1 c1 cours 342 Oct 18 15:28 fichA
prompt>
prompt> chmod g-rw fichA; ls -l fichA
-rwx---r-- 1 c1 cours 342 Oct 18 15:28 fichA
prompt>
prompt> chmod ug=rw fichA; ls -l fichA
-rw-rw-r-- 1 c1 cours 342 Oct 18 15:28 fichA
prompt>
```

Changer les droits d'un fichier ou d'un répertoire

Le code peut également être donné en mode octal.

Donc

775 (111 111 101) représente **rwx rwx r-x**

640 (110 100 000) représente **rw- r-- ---**

lettres	binaire	octal
---	000	0
--x	001	1
-w-	010	2
-wx	011	3
r--	100	4
r-x	101	5
rw-	110	6
rwx	111	7

Exemple :

```
prompt> ls -l toto
-rw-r--r-- 1 c1 cours 342 Oct 18 15:28 toto
prompt>
prompt> chmod 760 toto; ls -l toto
-rwxrw---- 1 c1 cours 342 Oct 18 15:28 toto
prompt>
```

Changer les droits d'un fichier ou d'un répertoire

- **umask : Modifie le masque des droits de création de fichier**

umask [masque]

Lorsqu'un programme crée un fichier, il spécifie les droits d'accès. Parmi ceux-ci, certains sont accordés, d'autres refusés, en fonction du masque. Sans argument, donne la valeur actuelle du masque. C'est le même pour les fichiers et les répertoires !

Exemple d'utilisation : prompt> umask 022

Pour les répertoires :				
droits demandés :	rwx	rwx	rwx	ou encore 777
- masque :	---	-w-	rwx	ou encore 027
droits accordés :	rwx	r-x	---	ou encore 750
Pour les fichiers :				
droits demandés :	rw-	rw-	rw-	ou encore 666
- masque :	---	-w-	-w-	ou encore 022
droits accordés :	rw-	r--	r--	ou encore 644

Changer le masque pour un fichier ou un répertoire

Exemple :

```
prompt> umask
000
prompt> mkdir foo
prompt> ls -ld foo
drwxrwxrwx 2 c1 cours 512 Jul 3 16:39 foo
prompt> rmdir foo
prompt> umask 022
prompt> mkdir foo
prompt> ls -ld foo
drwxr-xr-x 2 c1 cours 512 Jul 3 16:41 foo
prompt> rmdir foo
prompt> umask 077
prompt> mkdir foo
prompt> ls -ld foo
drwx------ 2 c1 cours 512 Jul 3 16:42 foo
prompt>
```

Recherche dans les fichiers

- **grep : recherche dans un ou plusieurs fichiers les lignes qui correspondent à un motif (chaine de caractères)**

  grep [options] motif chemin

Avec :

options: les options possibles (voir man grep)

recherche : le terme à rechercher entre guillemets

chemin : le chemin du fichier (ou dossier) où faire la recherche

Les guillemets autour du terme à rechercher ne sont pas obligatoire. Néanmoins je les conseils dès lors qu'il y a des caractères autres qu'alphanumériques dans votre recherche..

Exemple d'utilisation :

```
prompt> grep "^m" /etc/passwd
prompt> grep "192" /etc/hosts
grep "[0-9]\{10\}" * # recherche des suites de 10 chiffres dans tous les fichiers
```

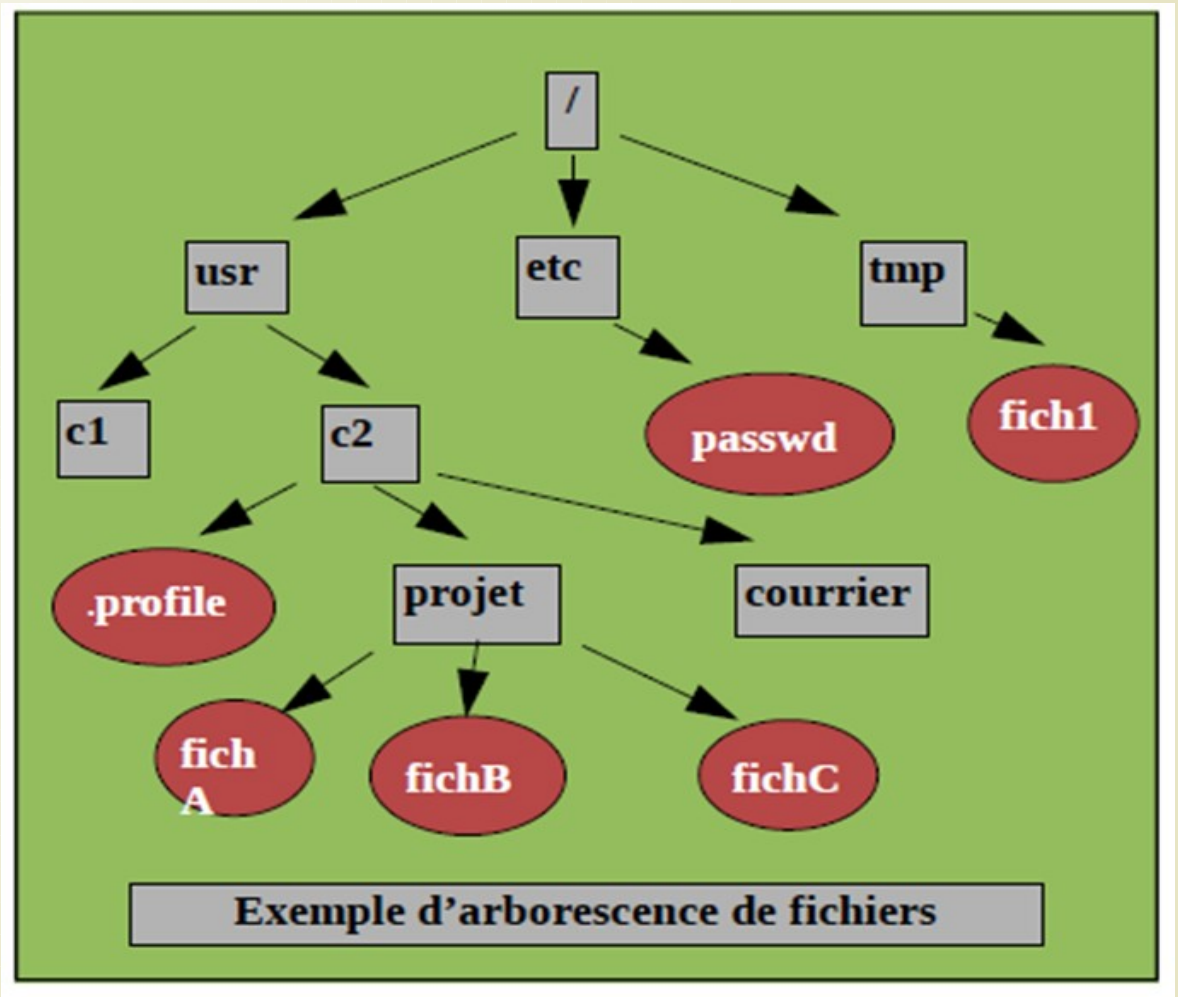

https://wodric.com/commande-grep/

Les répertoires :

Un ***répertoire*** ou ***catalogue (directory)*** est un **fichier** qui contient une liste de noms de fichiers, parmi lesquels on peut trouver des sous-répertoires, et ainsi de suite (***arborescence logique***).

Dans l'exemple ci-contre, les répertoires sont représentés par des rectangles et les fichiers par des cercles.

- ***Commandes associées***
- **mkdir** *répertoire :* [**m**a**k**e **dir**ectory] : crée un répertoire.
- **rmdir** *répertoire :* [**r**e**m**ove **dir**ectory] : détruit le répertoire s'il est vide et si ce n'est pas votre répertoire courant.
- **cd** *répertoire :* [**c**hange **d**irectory] : change de répertoire courant. Sans argument rapatrie dans le répertoire de connexion.
- **pwd :** [**p**rint **w**orking **d**irectory] : affiche le chemin absolu du répertoire courant.



Exemple d'arborescence de fichiers

Les répertoires :

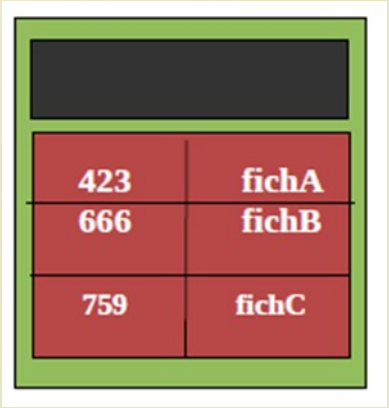
Un répertoire est un fichier ordinaire dans le sens où il possède un index et des données sous forme de suite d'octets. Seul le système peut écrire dans un répertoire.

répertoire = table de liens (i-nombre, chaîne de caractère).

i-nombre = numéro d'index (i-node) de la structure décrivant le fichier.

On peut voir les numéros d'i-nodes par la commande :

```
prompt> ls      -i
423     fichA
666     fichB
759     fichC
prompt>
```



Exemple de Fichier-répertoire "projet"

Manipulations de répertoires :

- **mkdir : Créer un répertoire**

Exemple d'utilisation :

```
prompt> mkdir rep
```

- **rmdir : Supprime un répertoire**

Exemple d'utilisation :

```
prompt> rmdir rep
```

on ne peut pas supprimer un répertoire vide

```
prompt> rmdir rep1
rmdir : 'rep' : Le répertoire n'est pas vide.
```

- **cp : Copier un répertoire, un fichier**

Exemple d'utilisation :

```
prompt> cp rep rep1
prompt> cp fichier fichier1
```

Manipulations de répertoires :

- **Copier une arborescence**

```
prompt> ls -R /tmp/r1
/tmp/r1 :
f1 f2 r11 r12
/tmp/r1/r11 :
f11
/tmp/r1/r12 :
f12
prompt> cp -r /tmp/r1 .
prompt> ls -R r1 r1 :
f1 f2 r11 r12
r1/r11 :
f11 r1/r12 :
f12
```


- **Supprimer une arborescence :** Attention il n'y a pas de demande de confirmation !

```
prompt> rm -rf r1
prompt> ls r1
ls : r1 : Aucun fichier ou répertoire de ce type.
```

Manipulations de répertoires :

- **Connaître la taille d'une arborescence et de chacun de ses sous-répertoires**

prompt> du .

8 ./.kde/Autostart

8 ./.kde

4 ./rep

56 .

- **Connaître le total (-s) avec la taille exprimée en K, M et G (-h)**

prompt> du -hs /home

620 M /home

Les chemins :

- ***Chemin absolu*** : Un chemin absolu se base sur la racine de l'arborescence et commence par / ou ~,

  ex. : `/home/utilisateur/<dossier>/<fichier>`.

  On accède au fichier à partir de la racine et en indiquant tous les sous-répertoires rencontrés jusqu'au fichier.

- ***Répertoire courant*** : A chaque processus est associé un répertoire, appelé répertoire courant ou de travail (défaut).

- ***Raccourcis Shell*** : Tilde **~,** utilisé en premier nom de répertoire, remplace le chemin absolu par son répertoire personnel, soit /home/utilisateur, mais cette fonctionnalité est propre au shell, et pas au système de fichier.

- ***Commandes associées***

  ln *fichier1 fichier2 :* [**l**i**n**k] :établit un nouveau lien sur le fichier *fichier1*.

Gestion des utilisateurs :

- **groupadd nom_groupe : Ajoute un groupe d'utilisateurs.**

Exemple d'utilisation :

```
prompt> groupadd but1a
```

- **groupdel nom_groupe : Détruit un groupe d'utilisateurs.**

Exemple d'utilisation :

```
prompt> groupdel but1a
```


- **useradd nom_utilisateur : Ajoute un utilisateur**

Exemple d'utilisation :

```
prompt> useradd jfa
```

- **usermod nom_utilisateur : Modifie les paramètres d'un compte utilisateur**

Exemple d'utilisation :

```
prompt> usermod -c "This is the best user" jfa
prompt> usermod -d /home/jfahome jfa
prompt> usermod -e 2020-05-29 jfa
```

https://www.geeksforgeeks.org/usermod-command-in-linux-with-examples/

Gestion des utilisateurs :

- **id nom_utilisateur : Affiche les informations d'identité d'un utilisateur.**

Exemple d'utilisation :

```
prompt> id jfa
```

- **groupdel nom_groupe : Détruit un groupe d'utilisateurs.**

Exemple d'utilisation :

```
prompt> groupdel but1a
```


- **useradd nom_utilisateur : Ajoute un utilisateur**

Exemple d'utilisation :

```
prompt> useradd jfa
```

- **usermod nom_utilisateur : Modifie les paramètres d'un compte utilisateur**

Exemple d'utilisation :

```
prompt> usermod -c "This is the best user" jfa
prompt> usermod -d /home/jfahome jfa
prompt> usermod -e 2020-05-29 jfa
```

https://www.geeksforgeeks.org/usermod-command-in-linux-with-examples/

Les chemins :

- ***Chemin relatif*** : c'est relatif au répertoire courant où se trouve l'utilisateur. Un chemin qui commence par autre chose que / ou ~ est un chemin relatif. Sous Linux on se trouve par défaut dans son répertoire personnel qui est /home/<nom d'utilisateur>. Dans un terminal on peut naviguer d'un répertoire à l'autre avec la commande **cd**.
- On peut aussi utiliser ce type de chemin pour indiquer où se trouvent les ressources les unes par rapport aux autres, indépendamment de la racine du système, par ex. pour que les fichiers d'un site web puissent se retrouver les uns les autres.
- On peut ne spécifier qu'une partie du chemin d'accès, pour que ce chemin soit interprété à partir du répertoire courant.

« . » : répertoire courant

« .. » : répertoire parent

Donc on peut désigner un fichier de 2 façons :

nom relatif ou **nom absolu**.

https://doc.ubuntu-fr.org/chemins

- ***cd .*** : avec la commande **cd** on se déplace dans le répertoire courant, donc on reste où on est !
- ***cd ..*** : avec la commande **cd** on se déplace dans le répertoire parent, donc le répertoire juste au dessus de là où on est !
- ***cd ~*** : avec la commande **cd** on se déplace dans le répertoire Home, donc le répertoire de login de l'utilisateur !

Gestion des liens entre fichiers

Un fichier (entête plus contenu) peut avoir plusieurs noms ; dans tous les cas, le système identifie un fichier non pas par un nom, mais par son numéro de i-noeud.

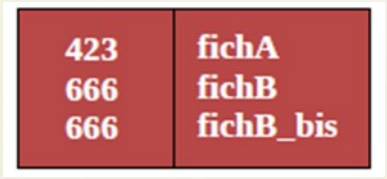
Un répertoire est un fichier dont le contenu contient un ensemble de couples **(nom, i-noeud)** ; **chaque nom est un lien vers un i-noeud donné**.

- ***Les liens "hard"***

Création de deux ou plusieurs noms vers un **i-noeud unique** au moyen de la commande :

ln fichB fichBbis

Cette commande n'est utilisable qu'à l'intérieur d'un même système de fichiers (arborescence).



prompt> **ls -i**
423 fichA
666 fichB
666 fichBbis
prompt>

prompt> **ls -ali**
389 drwxr-xr-x 2 c1 cours 342 Oct 08 15:27 .
125 drwxr-xr-x 5 c1 cours 342 Oct 08 15:27 ..
423 -rw-r--r-- 1 c1 cours 342 Oct 18 15:28 fichA
666 -rwxr-xr-x 2 c1 cours 106 Oct 10 10:53 fichB
666 -rwxr-xr-x 2 c1 cours 106 Oct 10 10:53 fichBbis
prompt>

Les liens "Symboliques"

Ces liens, différents des précédents, permettent de lier plusieurs noms à un même fichier sans rattacher ces derniers au i-noeud correspondant.

Pour cela on utilise la commande :

```
ln avec l'option -s
```

Cette commande est utilisable dans un système de fichiers (arborescence) ou entre des systèmes de fichiers différents.

La commande : `prompt> ln -s foo bar`

Crée le lien symbolique bar qui pointe sur le fichier foo.

```
prompt> ls -i bar foo
22195 bar
22192 foo
```

On a des i-noeuds différents. Par contre si l'on utilise la commande "ls -l" on voit apparaître le lien :

```
prompt> ls -l bar foo
lrwxrwxrwx 1 root root 3 Aug 5 16:51 bar->foo
-rw-r--r-- 1 root root 12 Aug 5 16:50 foo
prompt>
```


Les droits d'accès de la cible ("foo") via "bar" sont ceux de la cible.

Génération de noms de fichiers

Certains caractères spéciaux sont interprétés par le shell, et permettent de décrire les noms de fichiers. Ce sont *des méta-caractères* (c'est-à-dire des caractères utilisés pour décrire d'autres caractères) :

- **Le caractère '*'** signifie n'importe quelle chaîne de caractères.
- **Le caractère '?'** signifie n'importe quel caractère.
- **Les crochets '[ ]'** signifient un caractère appartenant à un ensemble de valeurs décrites dans les crochets.
- **Le caractère '-'** utilisé avec les crochets permet de définir un intervalle, plutôt qu'un ensemble de valeurs.
- **Le caractère '!' ou '^'** utilisé entre crochets en première position, signifie tout caractère excepté ceux spécifiés entre crochets.

Un ***nom générique*** est un mot qui contient un ou plusieurs caractères spéciaux. Il permet de désigner un ensemble d'objets.

Exemples avec

- **f*** Tous les fichiers dont le nom commence par 'f'.
- **f?** Tous les fichiers dont le nom commence par 'f', suivi d'un seul caractère quelconque.
- **f[12xy]** Tous les fichiers dont le nom commence par 'f', suivi d'un caractère à choisir parmi '1, '2', 'x' ou 'y'.
- **f[a-z]** Tous les fichiers dont le nom commence par 'f', suivi d'un caractère dont le code ASCII est compris entre le code 'a' et le code 'z', donc une lettre minuscule.
- ***.c** Tous les fichiers dont l'extension est .c
- **?.c** Tous les fichiers dont le nom est formé d'un caractère quelconque, suivi de '.c'
- **??** Tous les fichiers dont le nom est formé de deux caractères.
- ***.[A-Za-z]** Tous les fichiers dont le nom se termine par un '.' suivi d'une seule lettre majuscule ou minuscule.
- ***.[ch0-9]** Tous les fichiers dont le nom se termine par un '.' suivi d'un seul caractère à choisir parmi 'c', 'h', ou un chiffre entre '0' et '9'.
- **[!f]*** Tous les fichiers dont le nom ne commence pas par 'f'
- ***[!0-9]** Tous les fichiers dont le nom ne se termine pas par un chiffre.

Exemples avec les classes :

Pour définir les ensembles il est préférable d'utiliser la norme POSIX avec les noms de classe suivant :

- [:upper:] pour les majuscules
- [:lower:] pour les minuscules
- [:digit:] pour les chiffres de 0 à 9
- [:alnum:] pour les caractères alphanumériques

Ces noms s'utilisent de la façon suivante :

```
ls ./[[:upper:]]*
```

pour, par exemple, tous les noms de fichiers qui commencent par une lettre majuscule.

Substitution par le shell des méta-caractères :

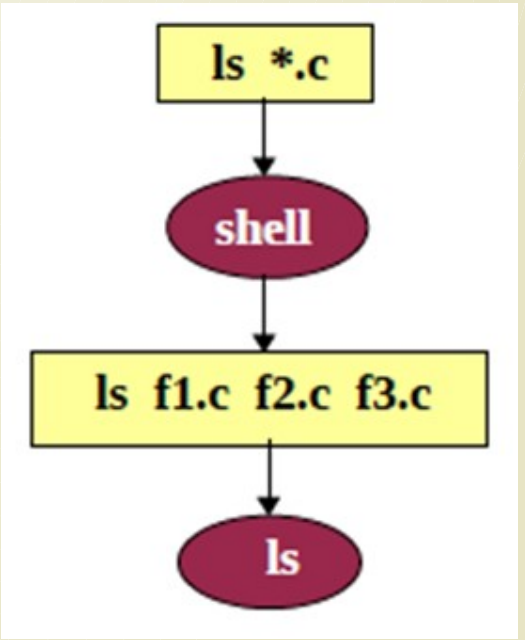
Lorsque le Shell trouve des métacaractères sur une ligne, il substitue l'expression entière, par exemple :

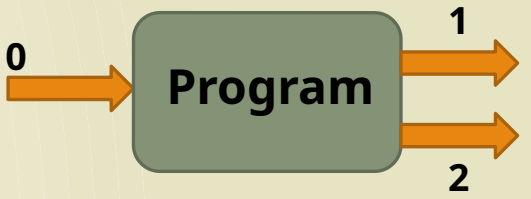
```
f* -> f1 f36 foo fz
```

Chaque ligne est donc analysée deux fois :

1. **Par le Shell** : celui-ci lit la ligne « ls -l f*.c » . Il enlève de cette ligne « f*.c » et le remplace par « f1.c f2.c f15.c ». Le résultat est donc la ligne « ls -l f1.c f2.c f15.c ».
2. **Par la commande** : lorsque celle-ci est exécutée, elle reçoit la ligne « ls -l f1.c f2.c f15.c », elle va donc travailler sur les 3 fichiers.

Le traitement des méta-caractères est indépendant de la commande. Il est effectué par le Shell, avant l'exécution de la commande.





Linux
Les bases du système

JFA - 139

DUT Informatique – Semestre 1

Ressource R1.04

Responsable : Jean-François ANNE

Les redirections d'entrées - sorties :

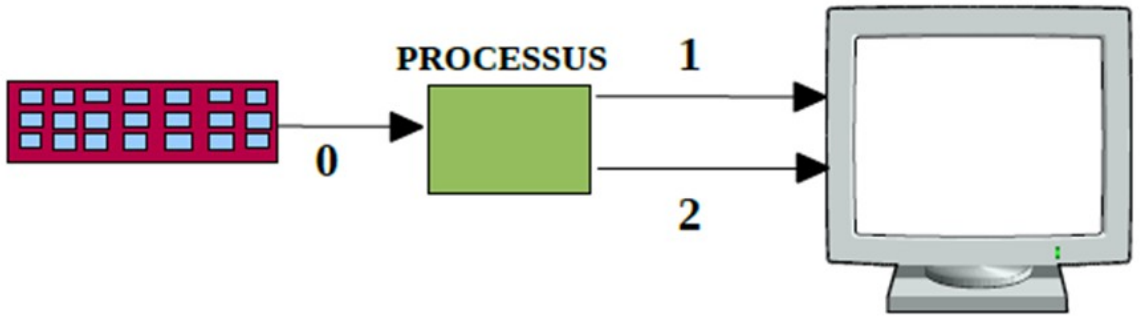
Tout processus effectue ses entrées/sorties via des « descripteurs de fichiers ». Ils sont numérotés à partir de 0.

Par convention une commande ou un programme utilise :

- **le descripteur 0** (entrée standard) pour lire des données,
- **le descripteur 1 (sortie standard) pour envoyer ses données en sortie,**
- **le descripteur 2 (sortie d'erreur) pour envoyer ses messages d'erreurs.**

Chaque descripteur peut être associé :

- à un fichier,
- à un périphérique (terminal par exemple),
- au descripteur d'un autre processus.



La configuration de ces descripteurs, c'est-à-dire leur association à un fichier ou à un périphérique, fait partie de l'environnement d'un processus.

Note : La commande elle-même ne connaît pas la destination finale des données qu'elle écrit. Elle se contente d'écrire sur un descripteur. C'est l'interpréteur de commandes qui met en place un environnement spécifique avant d'appeler la commande. Les demandes de redirection sont traitées par l'interpréteur de commandes avant même le lancement des commandes.

La redirection de la sortie standard :

Pour rediriger la sortie standard vers un fichier, vous devez utiliser la syntaxe suivante :

```
commande > fichier
```

Si le fichier n'existe pas, il est créé par le Shell. S'il existe déjà, le Shell détruit son contenu pour le remplacer par la sortie de la commande.

Exemple :

```
prompt> who > who.t
prompt>
```

Cette ligne ne produit aucune sortie sur le terminal. Toutes les informations de la commande ***who*** sont écrites dans le fichier ***who.t***

```
prompt> cat who.t
c1 tty1 Apr 25 02:44
c2 tty3 Apr 25 02:55
c3 tty6 Apr 25 02:33
prompt>
```

Le fichier ***who.t*** contient la sortie de la commande précédente ***who***, c'est-à-dire la liste des personnes connectées au système à l'instant où cette commande a été exécutée.

La redirection de la sortie standard :

Il est possible de rediriger plus d'une commande dans un fichier, il suffit de délimiter ces commandes par des parenthèses et séparées par des points virgules,

```
prompt> (date; who) > who.x
prompt>
```

Les sorties ***date*** et ***who*** seront redirigées dans le même fichier ***who.x***

Si on omet les parenthèses, le Shell exécute ***date***, imprime le résultat sur le terminal et ensuite lance ***who***, en redirigeant le résultat sur le fichier ***who.x.***

La redirection de la sortie standard :

Pour rediriger la sortie standard dans un fichier, vous pouvez également utiliser la syntaxe suivante :

```
commande >> fichier
```

Dans ce cas, la sortie de la commande est rajoutée à la fin de fichier. Comme précédemment, si fichier n'existe pas, le Shell le crée.


```
prompt> date > date.t ; cat date.t
Sun Jul 31 10:45:21 MET 1996
prompt> date >> date.t ; cat date.t
Sun Jul 31 10:45:21 MET 1996
Sun Jul 31 10:45:22 MET 1996
prompt>
```

La redirection de l’ entrée standard :

Le Shell permet aussi de rediriger l'entrée standard d'un processus (clavier), en utilisant la syntaxe suivante :

```
commande < fichier
```

Dans ce cas, la commande lit ses données dans un *fichier.*

```
prompt> cat < who.x
```


Cette commande affiche le contenu du fichier *who.x*.

Syntaxe générale :

- **n < fichier** redirige en lecture le descripteur *n* sur ***fichier***. Si ***n*** n'est pas précisé, ce sera 0 (l'entrée standard). Si ***fichier*** n'existe pas le shell renvoie une erreur.
- **n > fichier** redirige en écriture le descripteur ***n*** sur ***fichier***. Si ***n*** n'est pas précisé, ce sera 1 (la sortie standard). Si ***fichier*** existe il sera écrasé, sinon créé.
- **n >> fichier** redirige en écriture le descripteur ***n*** à la fin de ***fichier***, sans détruire les données préalablement contenues par ce fichier. Si ***n*** n'est pas précisé, ce sera 1 (la sortie standard). Si ***fichier*** n'existe pas il sera créé.

- **n < &m** duplique le descripteur ***n*** sur le descripteur ***m*** en lecture, ainsi ***n*** et ***m*** seront dirigés vers le même fichier, ou le même périphérique.
- **n > &m** duplique le descripteur ***n*** sur le descripteur ***m*** en écriture.
- **n < &-** ferme le descripteur ***n*** en lecture.
- **n > &-** ferme le descripteur ***n*** en écriture.

Syntaxe générale :

Exemple :

```
prompt> ls > f1 2> f2
```

Le listing du répertoire actuel est envoyé vers le fichier f1, qui si il n'existe pas est créé. Si il a une erreur à l'exécution du ls, cette erreur est envoyée dans le fichier f2, qui s'il n'existe pas est créé.

```
prompt> ls fich-inexistant >> f1 2>&1
```

Les caractéristiques du fichier-inexistant (si il existe) sont ajoutées à la fin du fichier f1. les erreurs sont dirigées vers le descripteur 1 qui est redirigé vers le fichier, donc les erreurs sont écrites à la fin du fichier f1,


```
prompt> ls fich-inexistant 1>> f1 2>>&1
```

Les caractéristiques du fichier-inexistant 1 (si il existe) sont affichées sur le descripteur 1 qui lui est redirigé vers le fichier f1, ajoutées à la fin. les erreurs sont dirigées vers le descripteur 1 qui est redirigé vers le fichier, donc les erreurs sont écrites à la fin du fichier f1,

```
echo 1234567890 > File    # Write string to "File".
exec 3<> File             # Open "File" and assign file desc. 3 to it.
read -n 4 <&3             # Read only 4 characters.
echo -n . >&3             # Write a decimal point there.
exec 3>&-                 # Close file desc. 3.
cat File                  # ==> 1234.67890
1234.67890
```

Double redirection :

Il est possible de rediriger à la fois l'entrée et la sortie.

Exemple :

```
prompt> wc < /etc/passwd > tmp
prompt> cat tmp
20 21 752
prompt>
```

- ✓ Les redirections d'entrées/sorties standards sont effectuées par le Shell d'une façon totalement indépendante du contexte. Les caractères spéciaux '>' et '<' peuvent être situés n'importe où sur une ligne de commande (précéder ou suivre la commande).

Les deux lignes suivantes produisent le même résultat :

```
prompt> who > tmp; grep 'c[12]' < tmp
c1 tty4 Jul 31 09:46
c2 tty2 Jul 31 09:17

prompt> > tmp who; < tmp grep 'c[12]'
c1 tty4 Jul 31 09:46
c2 tty2 Jul 31 09:17
prompt>
```

Double redirection :

- ✓ Les indications de redirections des entrées/sorties sont traitées par le Shell et ne sont pas passées à la commande. Le Shell appelle la commande après ces modifications.
- ✓ Un processus ne possède qu'une seule entrée, qu'une seule sortie et une seule sortie d'erreur. Par conséquent chaque descripteur ne peut être redirigé qu'une seule fois par commande !

Exemple :

```
prompt> cmd > fichier 1 > fichier 2
```

Cette ligne de commande n'a pas de sens, et *cmd verra sa sortie standard redirigée* uniquement vers *fichier2, après avoir* détruit le contenu de *fichier1.*

Attention :

Le Shell traite les séparateurs avant les redirections ; par conséquent il y a une différence importante entre les deux commandes suivantes :

Exemple :

```
prompt> cmd  2>  fichier

prompt> cmd  2  >  fichier
```


- Dans le premier cas, on redirige la sortie d'erreur de la commande cmd vers fichier.
- Alors que dans le second cas, on redirige la sortie standard de la commande cmd vers fichier et 2 sera considéré comme un argument de cmd (dû à l'espace entre le 2 et le >) !

Poubelle :

On peut rediriger la sortie d'erreur vers la « poubelle » (/dev/null); dans ce cas les messages d'erreur seront perdus :

Exemple :

```
prompt> cat fichier-inexistant 2>/dev/null
prompt>
```

Communication entre processus :

Les utilisateurs disposent d'un mécanisme leur permettant de lancer un certain nombre de processus de façon concurrente (en quelque sorte parallèlement) et communiquant entre eux par l'intermédiaire de tubes (pipes), le système assurant la synchronisation de l'ensemble des processus ainsi lancés.

Principe de fonctionnement : la sortie standard d'un des processus est redirigée sur l'entrée dans un tube et l'entrée standard de l'autre processus est redirigée sur la sortie de ce tube.

Ainsi les deux processus peuvent échanger des données sans passer par un fichier intermédiaire de l'arborescence.

Exemple :

```
prompt> cmd1 | cmd2
```

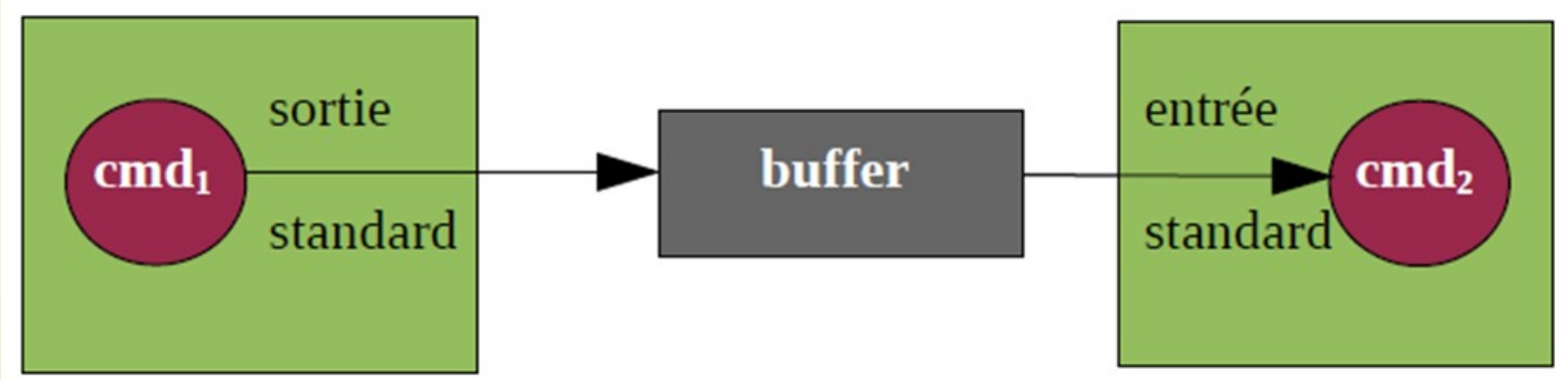


illustration de "cmd1 | cmd2"

Le lancement concurrent de processus communiquant deux par deux par l'intermédiaire d'un tube sera réalisé par une commande de la forme :

```
commande_1 | commande_2 | ... | commande_n
```

Exemple :

```
prompt> who > tmp ; grep cours < tmp
cours         ttya4  Jul 31 10:50
cours         ttyc6  Jul 31 09:34
cours         ttya2  Jul 31 09:02
prompt> rm tmp
prompt>
```

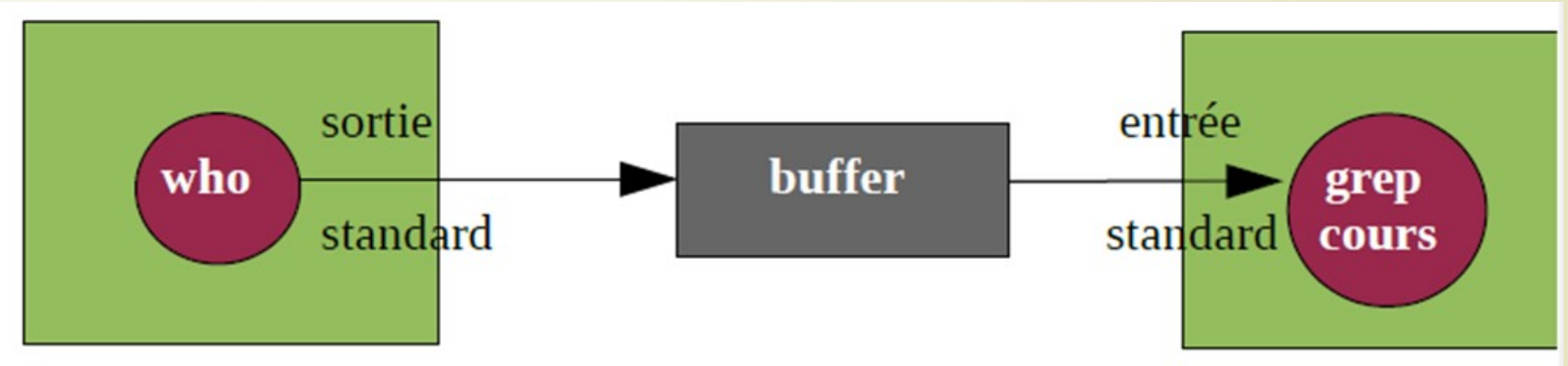
L'utilisation de la commande ***rm*** évite de conserver le fichier intermédiaire.

Communication entre processus :

L'utilisation d'un pipe (|) permet de résumer ces trois lignes de commandes en une seule :

```
prompt> who | grep cours
cours        ttya4         Jul 31 10:50
cours        ttyc6         Jul 31 09:34
cours        ttya2         Jul 31 09:02
prompt>
```

La sortie produite par la commande ***who*** est associée à l'entrée de la commande ***grep***. ***who*** donne la liste des personnes connectées au système à un moment donné ; ***grep*** cherche si la chaîne ***cours*** est présente dans le flot de données qu'elle reçoit. On peut donc considérer que la commande ***grep*** joue le rôle de filtre.



prompt> who | grep cours

Communication entre processus :

Examinons l'exemple suivant :

```
prompt> ps -a | wc -l
9
prompt>
```

La commande "**ps -a | wc -l**" entraîne la création de deux processus concurrents (allocation du processeur). Un tube (fichier particulier, appelé *buffer*) est créé dans lequel les résultats du premier processus ("**ps -a**") sont écrits. Le second processus lit dans le tube.

Lorsque le processus écrivain se termine et que le processus lecteur dans le tube a fini d'y lire (le tube est donc vide et sans lecteur), ce processus détecte une fin de fichier sur son entrée standard et se termine.

Le système assure la synchronisation de l'ensemble dans le sens où :

- il bloque le processus lecteur du tube lorsque le tube est vide en attendant qu'il se remplisse (s'il y a encore des processus écrivains);
- il bloque (éventuellement) le processus écrivain lorsque le tube est plein (si le lecteur est plus lent que l'écrivain et que le volume des résultats à écrire dans le tube est important).

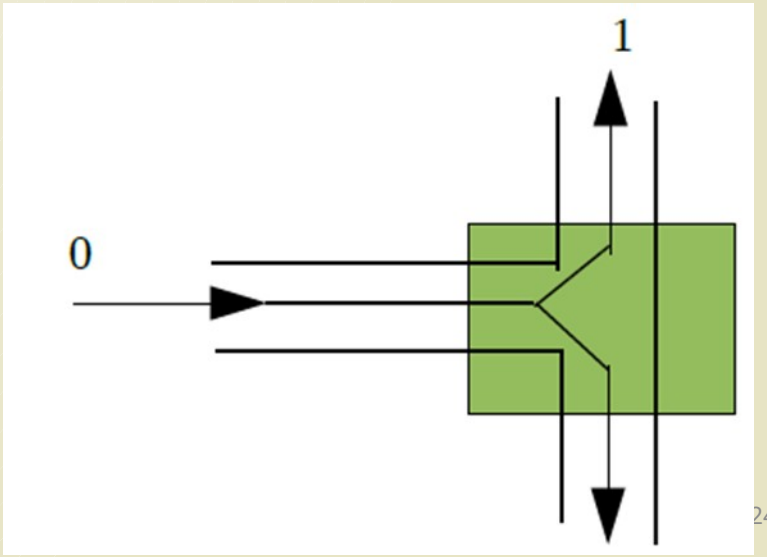
Le système assure l'implémentation des tubes. Il est chargé de leur création et de leur destruction.

Un tube de communication (|) permet de mémoriser des informations. Il se comporte comme une file FIFO, d'où son aspect unidirectionnel.

Pour vérifier que plusieurs processus existent simultanément, il suffit de lancer la séquence suivante :

```
prompt> ps -l | tee /dev/tty | wc -l
F S UID PID PPID PRI ... CMD
1 S 102 241 234 158 -bash
1 R 102 294 241 179 ps
1 S 102 295 241 154 tee
1 S 102 296 241 154 wc
5
prompt>
```


La commande "**tee fichier**" utilisée ici correspond au schéma suivant :
Elle permet d'envoyer à la fois sur sa sortie standard et dans le fichier de référence donné en paramètre ce qu'elle lit sur son entrée standard (ici la référence utilisée est ***/dev/tty*** qui permet de visualiser à l'écran les résultats de la commande **ps**).



fichier

PS : Les commandes ayant la propriété à la fois de lire sur leur entrée standard et d'écrire sur leur sortie standard sont appelées des **filtres**. Les commandes **cat**, **wc**, **sort**, **grep**, **sed**, **sh** ou **awk** sont des filtres ; par contre **echo**, **ls** ou **ps** n'en sont pas.

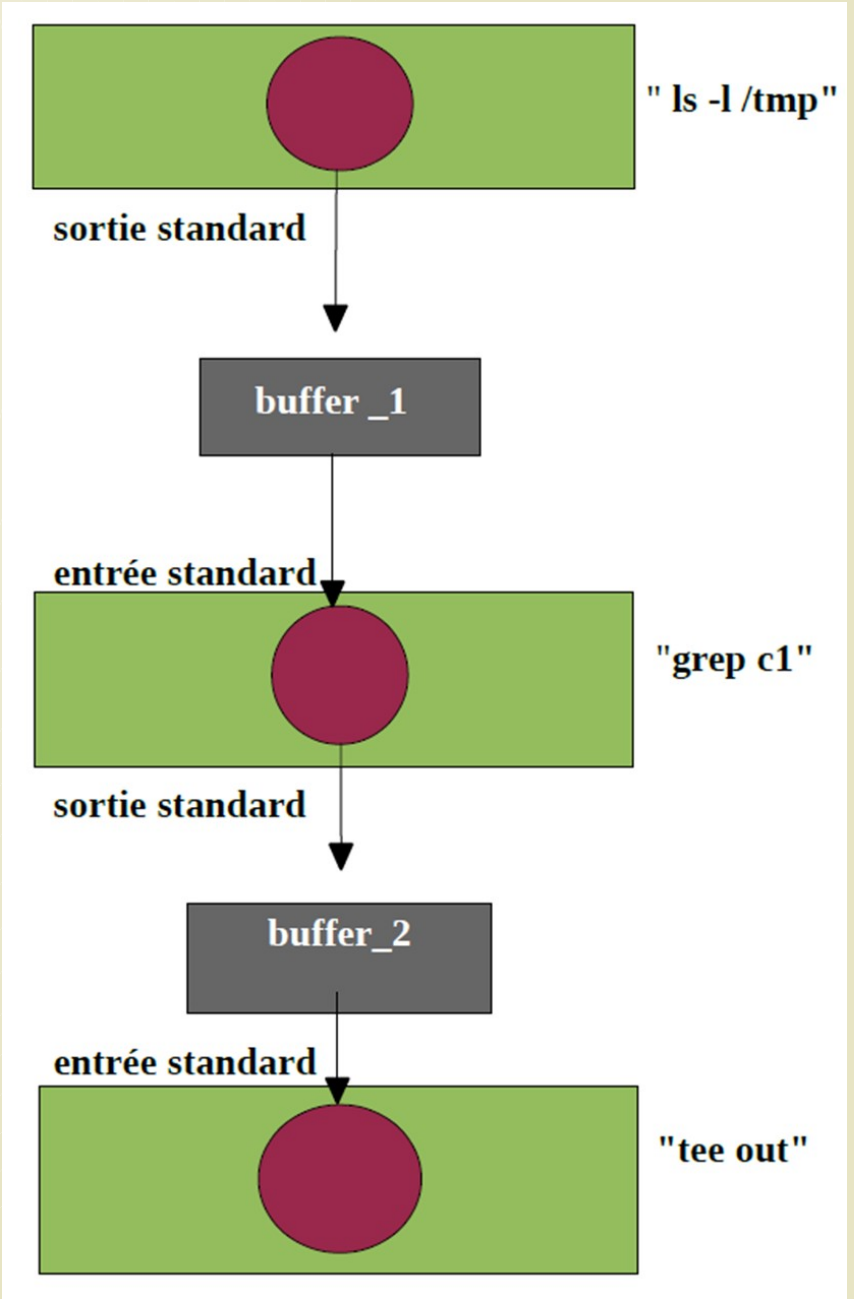
Autre exemple d'utilisation de la commande **tee** :

```
prompt> ls -l /tmp | grep c1 | tee out
-rw-------   1     c1    cours        37888          ...    Ex1
-rw-------   1     c1    cours        5120           ...    Ex2
prompt> cat out
-rw-------   1     c1    cours        37888          ...    Ex1
-rw-------   1     c1    cours        5120           ...    Ex2
prompt>
```


- **ls** envoie la liste des fichiers contenus dans le répertoire */tmp* sur sa sortie standard
- cette sortie est associée à l'entrée standard de **grep**
- **grep** lit les données qui arrivent sur son entrée standard, sélectionne les lignes contenant *c1*, et les envoie sur sa sortie standard
- cette sortie est associée à l'entrée standard de **tee**
- **tee** se contente de dédoubler ses sorties, c'est-à-dire de lire les données qui arrivent sur son entrée standard, et d'envoyer ces données sur sa sortie standard et sur le fichier dont le nom est donné en argument, ici « *out* ».

Communication entre processus :

Note : vous ne verrez aucun des résultats intermédiaires de cette suite de pipes.



Un exemple de suite de tubes pour le filtrage des données consiste à afficher l'espace libre (en kilo-octet) sur la partition qui contient le répertoire de travail.

```
prompt> df -k . | tail -1 | sed "s/ */ /g" | cut -d " " -f 4
60833156
prompt>
```

Nous allons détailler cette ligne de commande :

Affichage des statistiques en kilo-octet (option -k) sur le répertoire courant (.) :

```
prompt> df -k .
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/sda7 98123404 32281532 60833156 35% /
prompt>
```


On ne garde qu'une seule ligne en partant de la fin :

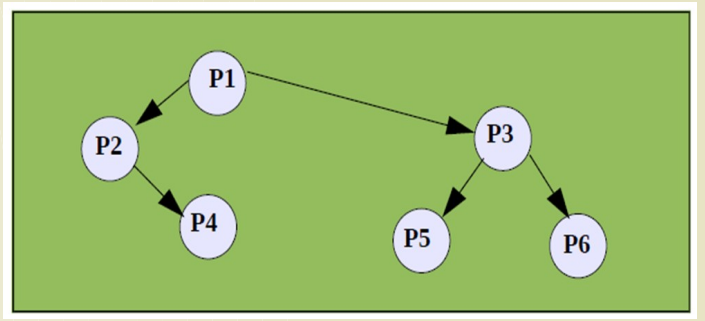
```
prompt> df -k . | tail -1
/dev/sda7 98123404 32281532 60833156 35% /
prompt>
```

On ne garde qu'un seul espace entre chaque mot :

```
prompt> df -k . | tail -1 | sed "s/ */ /g"
/dev/sda7 98123404 32281532 60833156 35% /
prompt>
```

On ne garde que le quatrième champ de la ligne, l'option « d » précise le séparateur à prendre en compte :

```
prompt> df -k . | tail -1 | sed "s/ * / /g" | cut -d " " -f 4
60833156
prompt>
```



Linux

Les processus

JFA - 160

DUT Informatique – Semestre 1

Ressource R1.04

Responsable : Jean-François ANNE

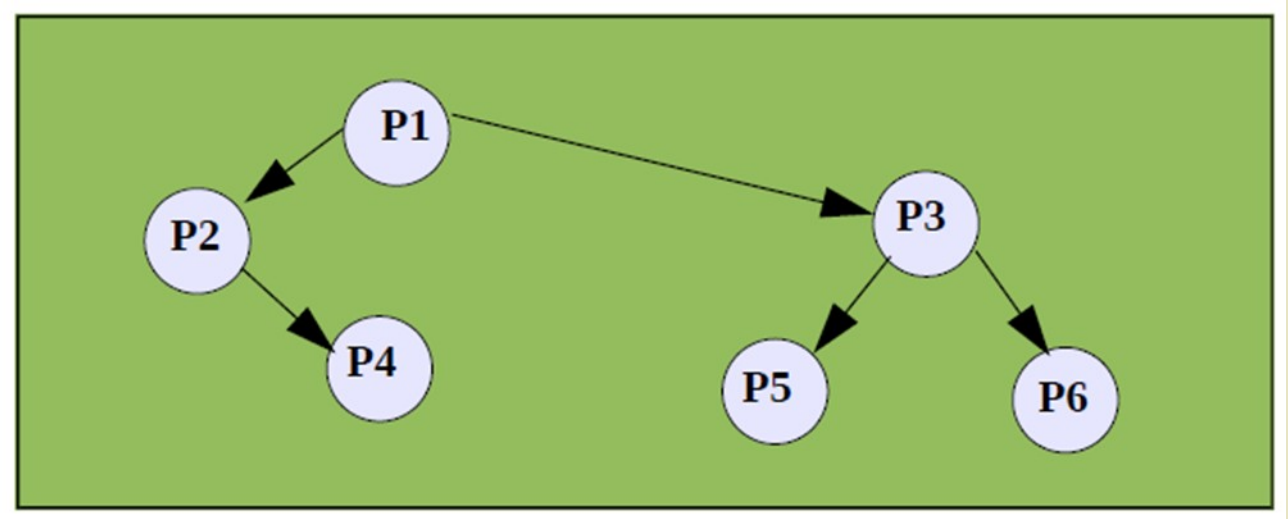
Introduction aux processus :

Un processus (process) est un programme (un ensemble d'octets en langage machine) en cours d'exécution dans un ordinateur.

Un système d'exploitation doit en général traiter plusieurs tâches en même temps. Comme il n'a, la plupart du temps qu'un processeur, il résout ce problème grâce à un pseudo-parallélisme. Il traite une tâche à la fois, s'interrompt et passe à la suivante. La commutation de tâches étant très rapide, il donne l'illusion d'effectuer un traitement simultané.

Les processus des utilisateurs sont lancés par un interprète de commande (Shell). Ils peuvent eux-mêmes lancer ensuite d'autres processus. On appelle le processus créateur, le père, et les processus créés, les fils. Les processus peuvent donc se structurer sous la forme d'une arborescence.

Au lancement du système, il n'existe qu'un seul processus, appelé processus «init» (P1), qui est l'ancêtre de tous les autres.

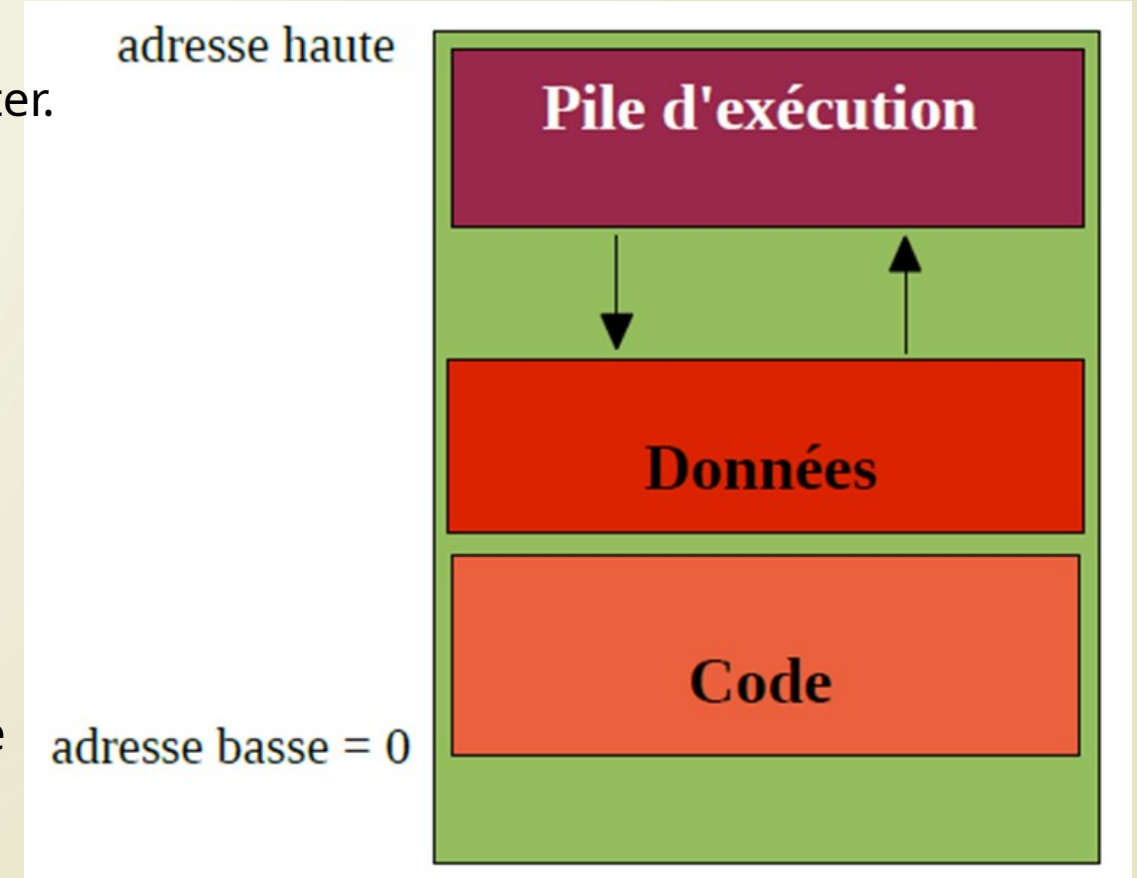


Introduction aux processus :

Les processus sont composés d'un espace de travail (espace d'adressage) en mémoire formé de 3 segments et visible par l'utilisateur/programmeur :

- Le code correspond aux instructions, en langage d'assemblage, du programme à exécuter.
- La zone de données contient les variables globales ou statiques du programme ainsi que les allocations dynamiques de mémoire.
- Enfin, les appels de fonctions, avec leurs paramètres et leurs variables locales, viennent s'empiler sur la pile.

Les zones de pile et de données ont des frontières mobiles qui croissent en sens inverse lors de l'exécution du programme.

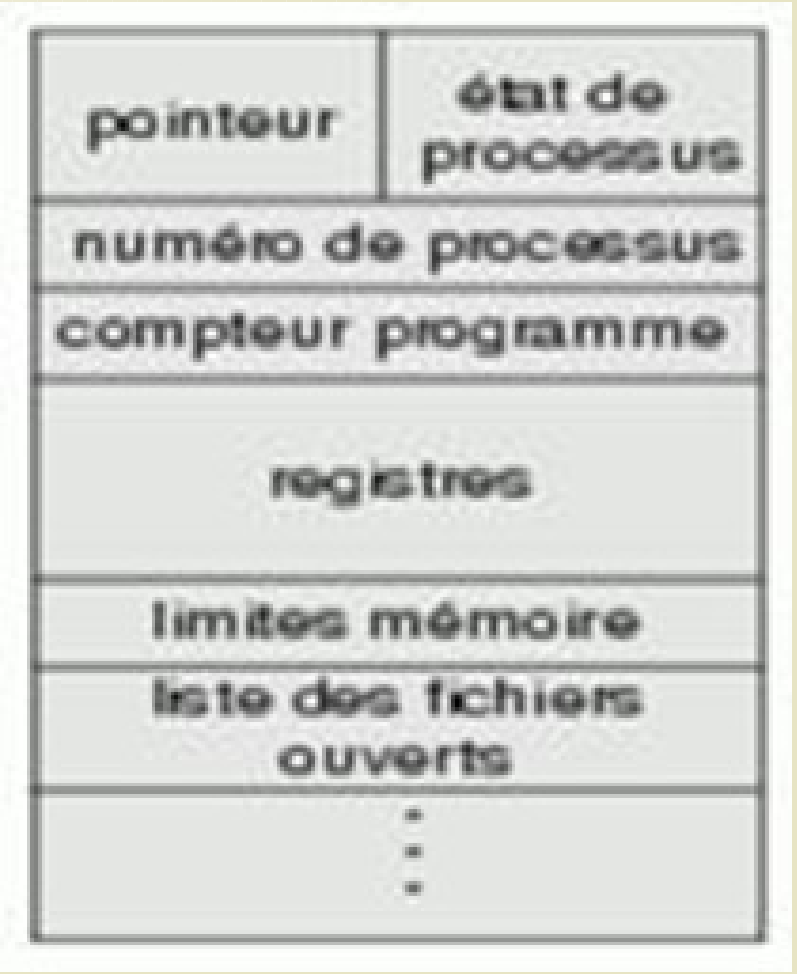


Le process Control Bloc :

Un processus est une entité active avec son propre compteur ordinal (*instruction pointer*) et l'ensemble des ressources qui lui sont associées. Ces informations sont stockées, pour chaque processus dans le PCB (*process control block*).

Le PCB va contenir typiquement :

- L'ID du processus (PID), l'ID du processus parent (PPID) et l'ID de l'utilisateur du processus (UID) ;
- Les valeurs des registres correspondant au processus (l'état courant du processus, selon qu'il est élu, prêt ou bloqué) ;
- Le compteur ordinal du processus ;
- Le pointeur de pile : indique la position du prochain emplacement disponible dans la pile mémoire ;
- L'espace d'adressage du processus ;
- La liste des descripteurs de fichiers ;
- La liste de gestion des signaux ;
- D'autres informations telles que le temps CPU accumulé par le processus, etc.



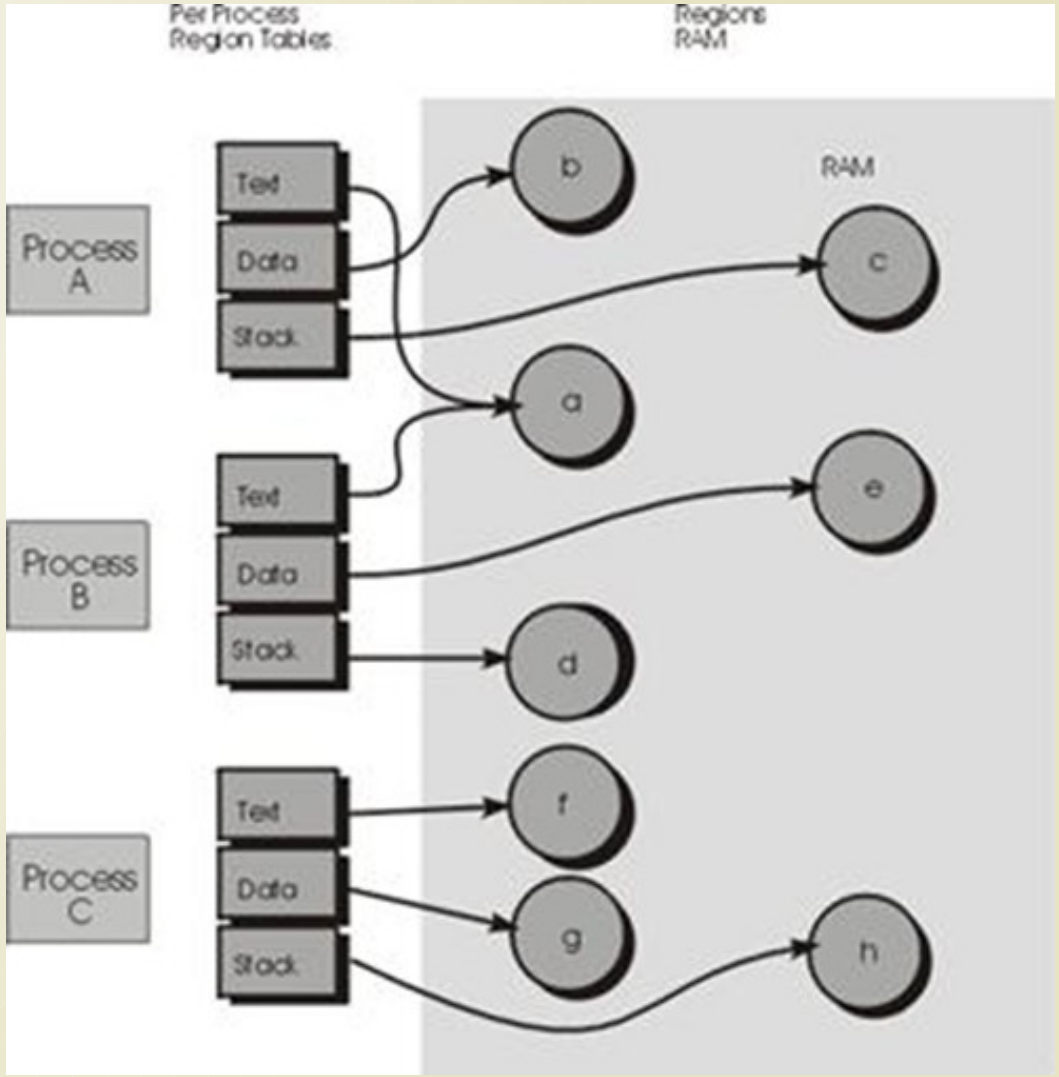
Process Control Bloc

Processus Vs Programme :

Lors d'un changement de contexte, le processus en cours est arrêté et un autre processus peut utiliser le CPU. Le noyau doit arrêter l'exécution du processus en cours, copier les valeurs des registres hardware dans le PCB, et mettre à jour les registres avec les valeurs du nouveau processus. Et lancer l'exécution du nouveau processus.

Un processus est un programme qui s'exécute et qui possède en plus, son compteur ordinal, ses registres et ses variables ; c'est là toute la subtilité entre programme et processus. La différence entre un processus et un programme est mince : le processus possède le programme mais également l'état courant de celui-ci dans la mémoire de l'ordinateur. Le programme est en fin de compte l'ensemble des fichiers qui, lorsqu'ils sont exécutés, deviennent le processus.

Un **processus** est une activité d'un certain type qui possède un **programme**, des données en entrée et en sortie, ainsi qu'un état courant.



Format d'un fichier exécutable

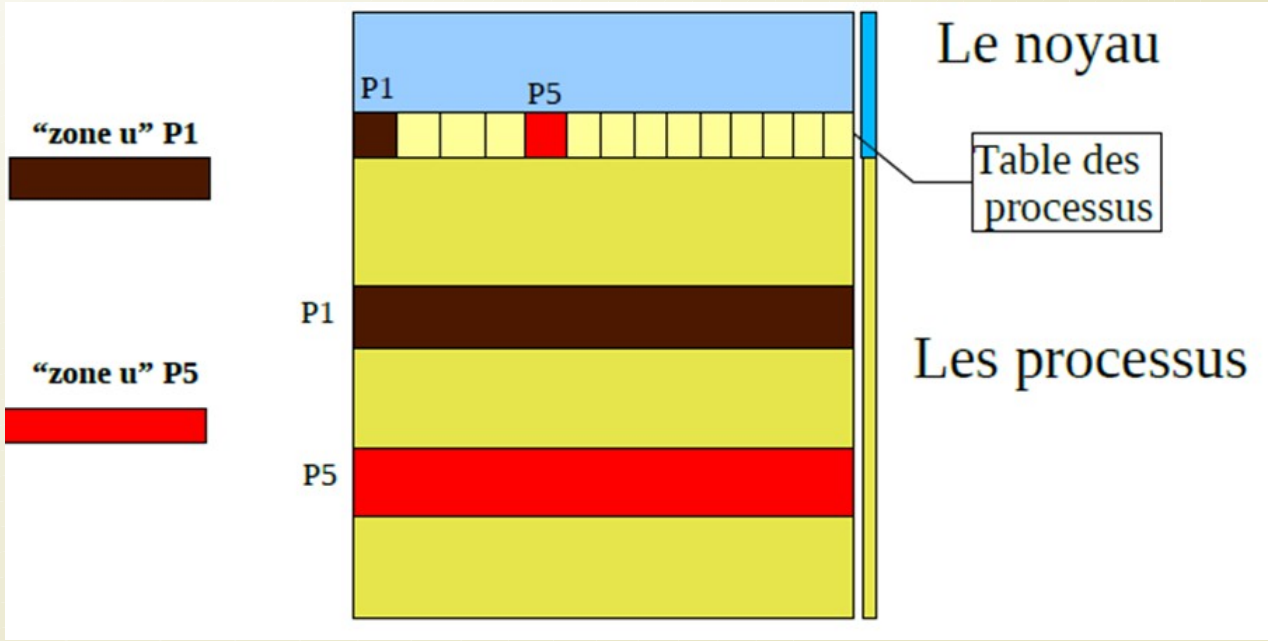
Processus Vs Programme :

Un processus est donc un «programme» qui s'exécute et qui possède :

- Son propre compteur ordinal (@ de la prochaine instruction exécutable),
- Ses registres et ses variables.

Le concept de processus n'a donc de sens que dans le cadre d'un contexte d'exécution. Conceptuellement, chaque processus possède son propre processeur virtuel, en réalité, le vrai processeur commute entre plusieurs processus (**multiprogrammation**).

Le noyau maintient une table, appelée « table des processus », pour gérer l'ensemble des processus (ici P1, ..., P5, ...). Cette table, interne au noyau, contient la liste de tous les processus avec des informations concernant chaque processus. C'est un tableau de structure « proc » (<sys/proc.h>).



La zone U :

Le nombre des emplacements dans cette table des processus est limité pour chaque système et pour chaque utilisateur. Le noyau alloue pour chaque processus une structure appelée « zone u » (<sys/user.h>), qui contient des données privées du processus, uniquement manipulables par le noyau.

Seule la « zone u » du processus courant est manipulable par le noyau, les autres sont inaccessibles.

L'adresse de la « zone u » d'un processus est placée dans son mot d'état.

Le noyau dispose donc d'un tableau de structures (« proc.h ») dans la table des processus et d'un ensemble de structures (« user.h »), une par processus, pour piloter les processus.

Le **contexte d'un processus** est l'ensemble des données qui permettent de reprendre l'exécution d'un processus qui a été interrompu :

son état (élu, prêt, bloqué, ...)
son mot d'état : en particulier
    * la valeur des registres actifs
    * le compteur ordinal
les valeurs des variables globales statiques ou dynamiques
son entrée dans la table des processus
sa « zone u »
les piles « user » et « system »
les zones de code (texte) et de données

Le noyau et ses variables ne font partie du contexte d'aucun processus. L'exécution d'un processus se fait dans son contexte.

Changement de processus :

Quand il y a changement de processus courant, il y a réalisation d'une commutation de mot d'état et d'un changement de contexte (multiprogrammation).

Le noyau s'exécute alors dans le nouveau contexte.

Ce passage d'une tâche à une autre, est réalisée par un ordonnanceur (scheduler) au niveau le plus bas du système. Cet ordonnanceur est activé par des interruptions d'horloge, de disque et de terminaux.

A chaque interruption correspond un vecteur d'interruption, c'est-à-dire un emplacement mémoire contenant une adresse. L'arrivée de l'interruption provoque le branchement à cette adresse.

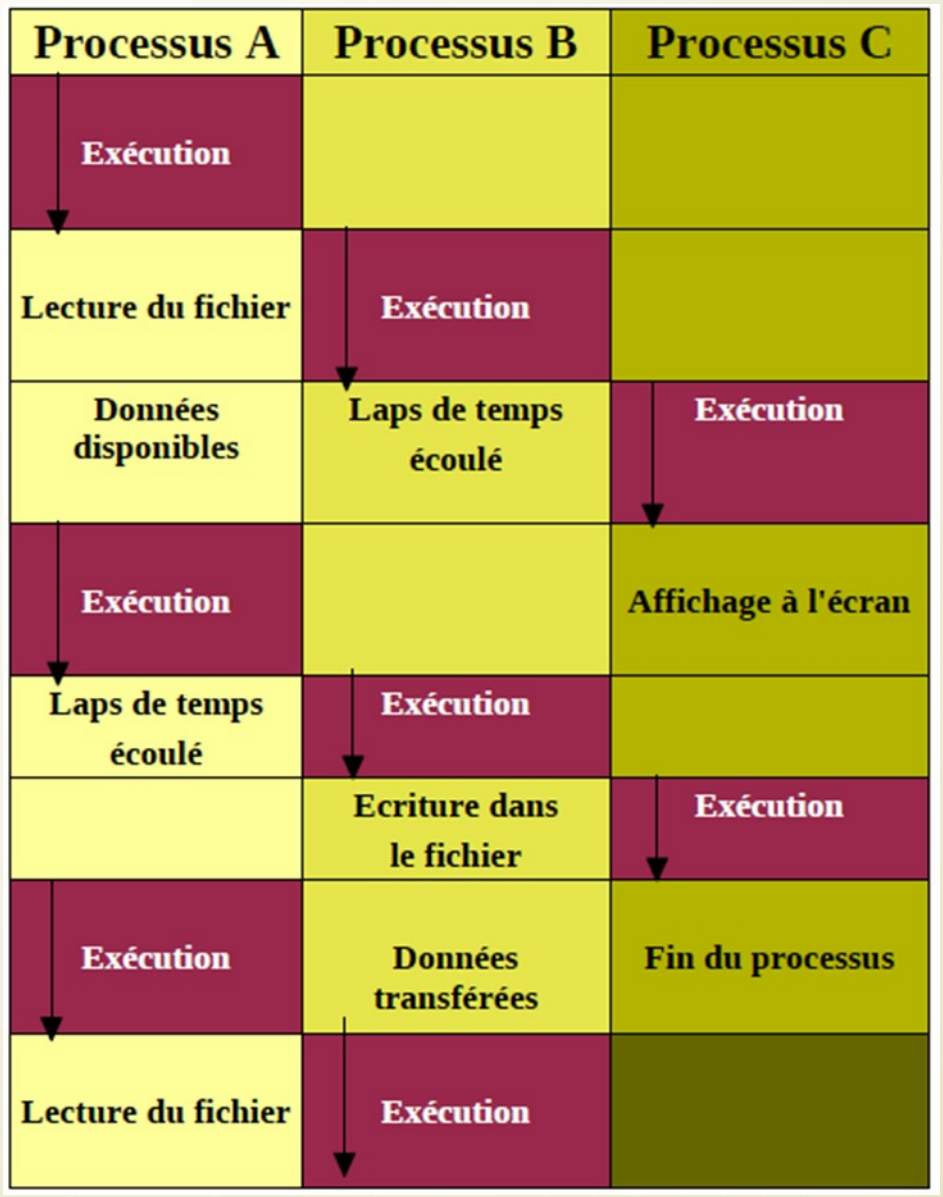
Sur un intervalle de temps assez grand, tous les processus ont progressé, mais à un instant donné un seul processus est actif.

Comme le processeur commute entre les processus, la vitesse d'exécution d'un processus ne sera pas uniforme et variera vraisemblablement si les mêmes processus étaient exécutés à nouveau.

Il ne faut donc pas que les processus fassent une quelconque présomption sur le facteur temps.

Sur le schéma ci-dessous, les trois programmes deviennent trois processus indépendants qui ont chacun leur propre contrôle de flux (compteur ordinal).

Processus :


Processus A
Processus B
Processus C
Exécution
Lecture du fichier
Exécution
Données disponibles
Laps de temps écoulé
Exécution
Exécution
Affichage à l'écran
Laps de temps écoulé
Exécution
Ecriture dans le fichier
Exécution
Exécution
Données transférées
Fin du processus
Lecture du fichier
Exécution
Temps

Processus :

Parmi les informations propres à chaque processus, qui sont contenues dans les structures système (« proc.h » et « user.h ») , on trouve :

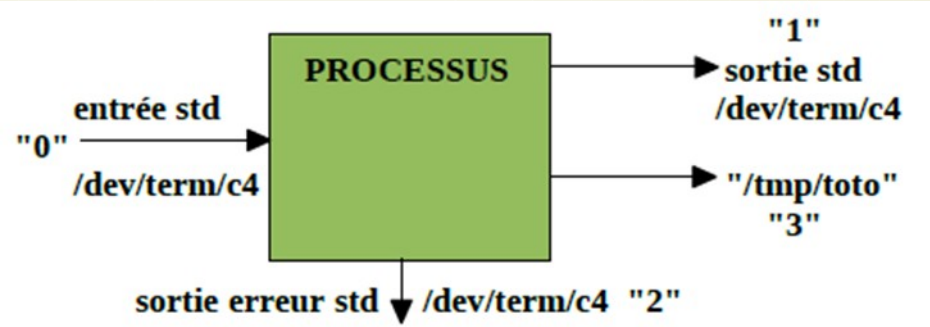
- un numéro d'identification unique appelé PID (Process Identifier), ainsi que celui de son père appelé PPID
- le numéro d'identification de l'utilisateur qui a lancé ce processus, appelé UID (User IDentifier), et le numéro du groupe auquel appartient cet utilisateur, appelé GID (Group IDentifier) ;
- le répertoire courant ;
- les fichiers ouverts par ce processus ;
- le masque de création de fichier, appelé umask ;
- la taille maximale des fichiers que ce processus peut créer, appelée ulimit ;
- la priorité ;
- les temps d'exécution ;
- le terminal de contrôle, c'est-à-dire le terminal à partir duquel la commande a été lancée.

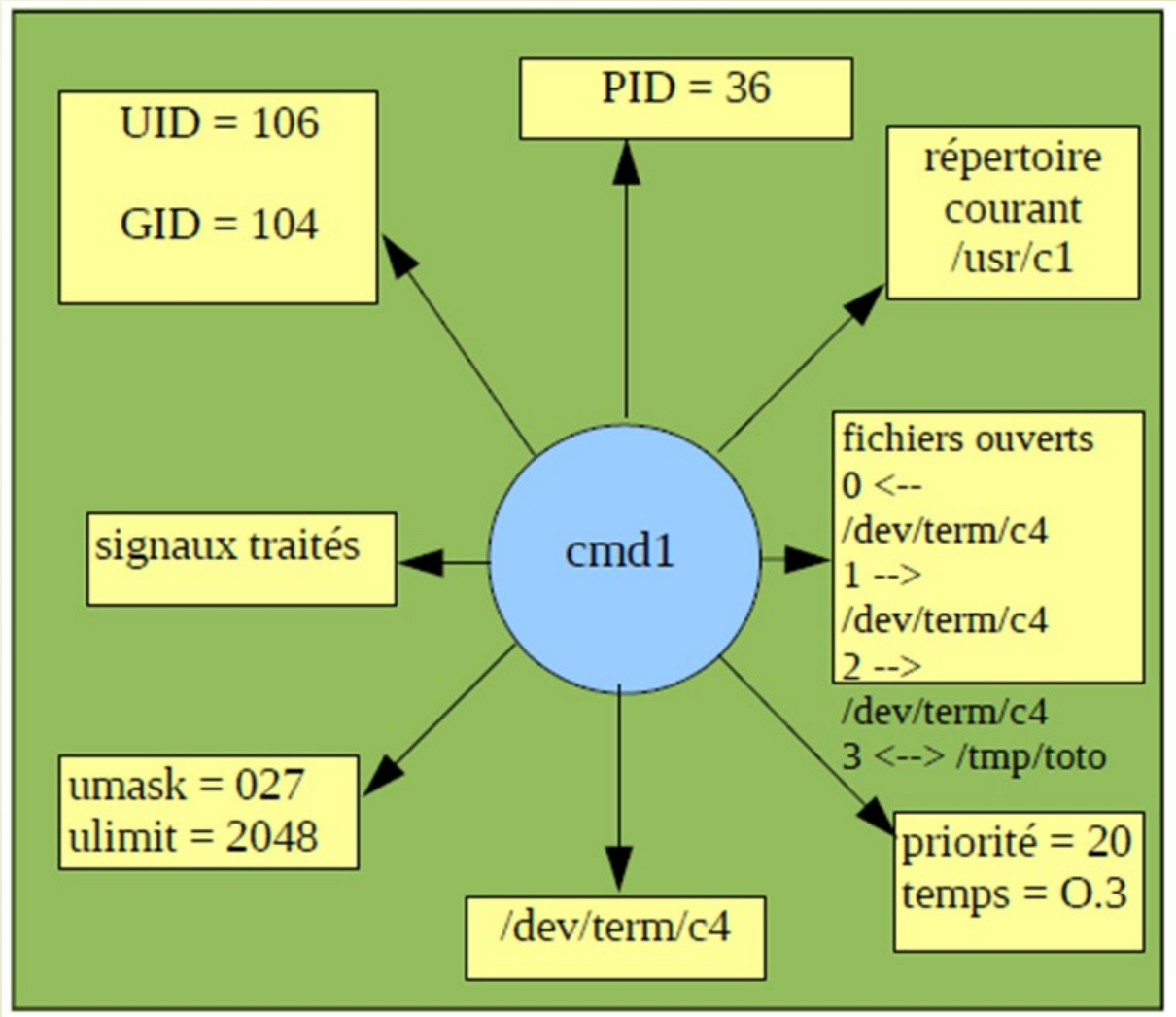
Processus :

Voici un premier schéma (simplifié) d'un processus :

Ce processus a le numéro *36*. Il a été lancé par l'utilisateur qui a *106* pour UID. Il est en train d'exécuter le programme '*cmd1'*. Il a *consommé 0.3* seconde, avec une priorité de *20*. Son masque de création de fichier est *027*. Son terminal de contrôle est */dev/term/c4*. Son répertoire courant est */usr/c1*.

Il a 4 fichiers ouverts :





Métaphore :

Une métaphore illustrant la différence entre processus et programme :

Soit un informaticien qui prépare un gâteau d'anniversaire pour sa fille.

- Il a une recette pour faire le gâteau et dispose de farine, d'oeufs, de sucre ...
- Ici la recette représente le programme (algorithme traduit en une suite d'instructions), l'informaticien joue le rôle du processeur (CPU) et les ingrédients sont les données à fournir.
- Le processus est l'activité de notre cordon bleu qui lit la recette, trouve les ingrédients nécessaires et fait cuire le gâteau.
- Si le fils de l'informaticien arrive en pleurant parce qu'il a été piqué par une guêpe, son père marque l'endroit où il était dans la recette (l'état du processus en cours est sauvegardé), cherche un livre sur les premiers soins et commence à soigner son fils.
- Le processeur passe donc d'un processus (la cuisine) à un autre plus prioritaire (les soins médicaux), chacun d'eux ayant un programme propre (la recette et le livre des soins).
- Lorsque la piqûre de la guêpe aura été soignée, l'informaticien reprendra sa recette à l'endroit où il l'avait abandonnée.

Encore quelques commandes :

Certaines des caractéristiques de l'environnement peuvent être consultées par diverses commandes. Nous connaissons déjà :

- **pwd** affiche le chemin du répertoire courant
- **tty** affiche le terminal de contrôle
- **umask** affiche le masque de création de fichier

Voici d'autres commandes :

- ***La commande* id**

id consulte l'UID et le GID.

- ***Exemple :***

```
prompt> id
uid=106(c1) gid=104(cours)
prompt>
```

- ***La commande* whoami**

whoami affiche uniquement le nom associé à l'UID.

- ***Exemple :***

```
prompt> whoami
 c1
prompt>
```

Encore quelques commandes :

❑ *La commande* pid

Le **PID** est stocké dans une pseudo-variable spéciale que l'on appelle « **$** ».

On peut consulter le PID du Shell courant en tapant :

❖ ***Exemple :***

```
prompt> echo $$
36
prompt>
```


Le 1er "**$**" définit le contenu de la pseudo- variable, alors que le second "**$**" correspond au PID du Shell courant.

Encore quelques commandes :

La commande find

find parcourt récursivement l'arborescence en sélectionnant des fichiers selon des critères de recherche, et exécute des actions sur chaque fichier sélectionné.

Syntaxe générale

```
find répertoire_de_départ critère_de_recherche action_à_exécuter
```

Exemple :

```
$ find $HOME -print
```


Cette commande va parcourir toute l'arborescence à partir du répertoire de login (**$HOME**), va sélectionner tous les fichiers puisqu'il n'y a aucun critère de recherche, et va afficher le nom de chaque fichier trouvé.

Le critère de recherche name

-name modèle sélectionne uniquement les fichiers dont le nom correspond au modèle.

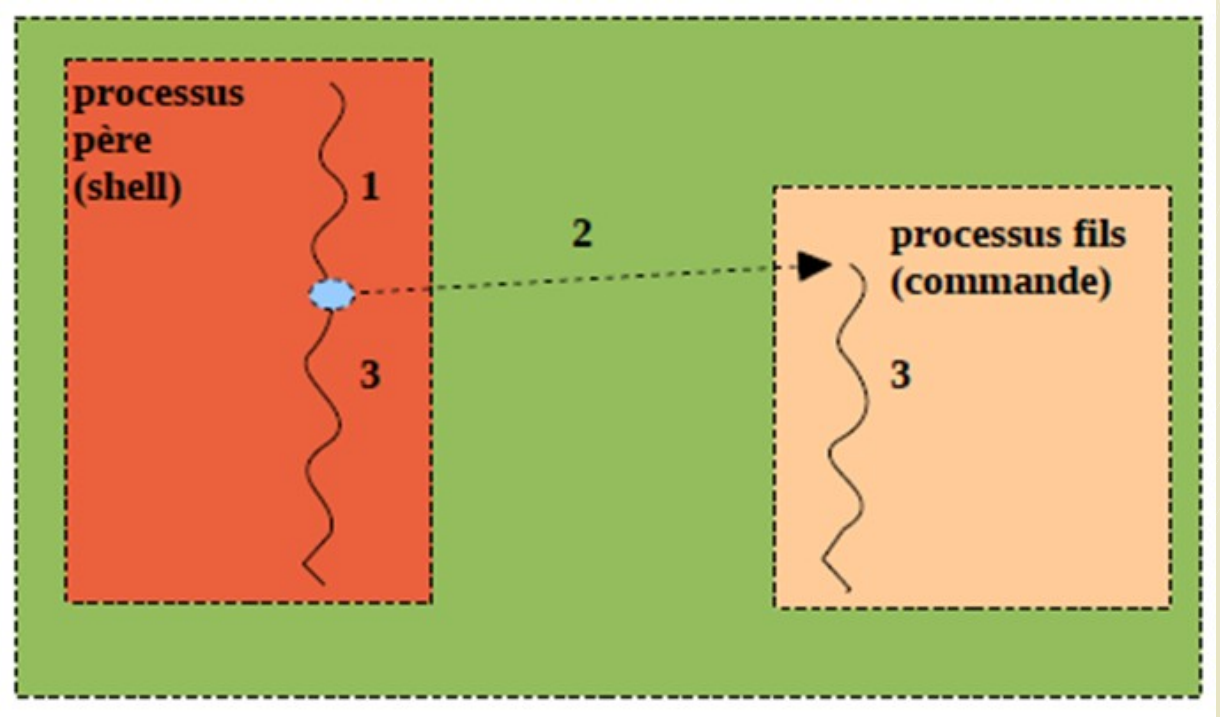
Attention ! Le modèle doit être interprété par la commande **find** et non par le shell, donc s'il contient des caractères spéciaux pour le shell (par exemple *), ceux-ci doivent être protégés.

Création de processus :

Pour chaque commande lancée (sauf les commandes internes), le Shell crée automatiquement un nouveau processus.

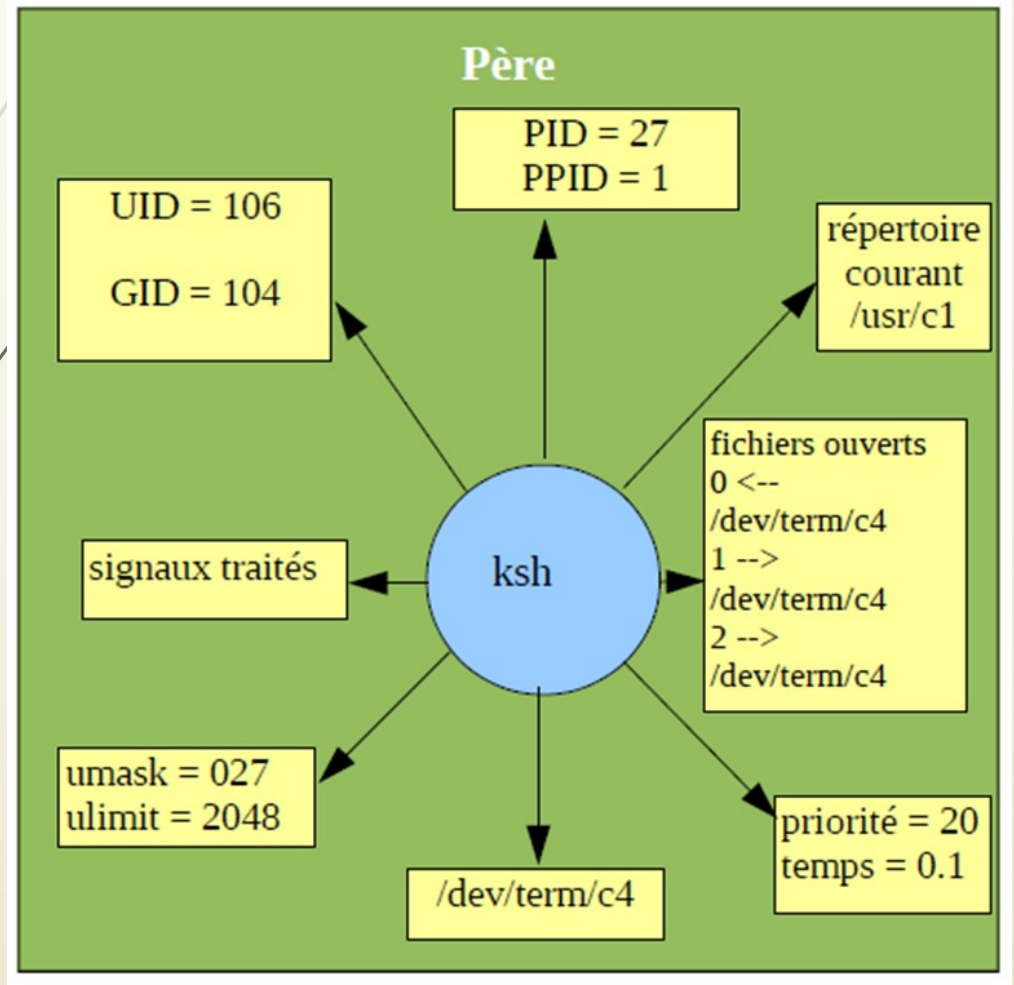
Il y a donc 2 processus. Le premier, appelé processus père, exécute le programme Shell, et le deuxième, appelé processus fils, exécute la commande.

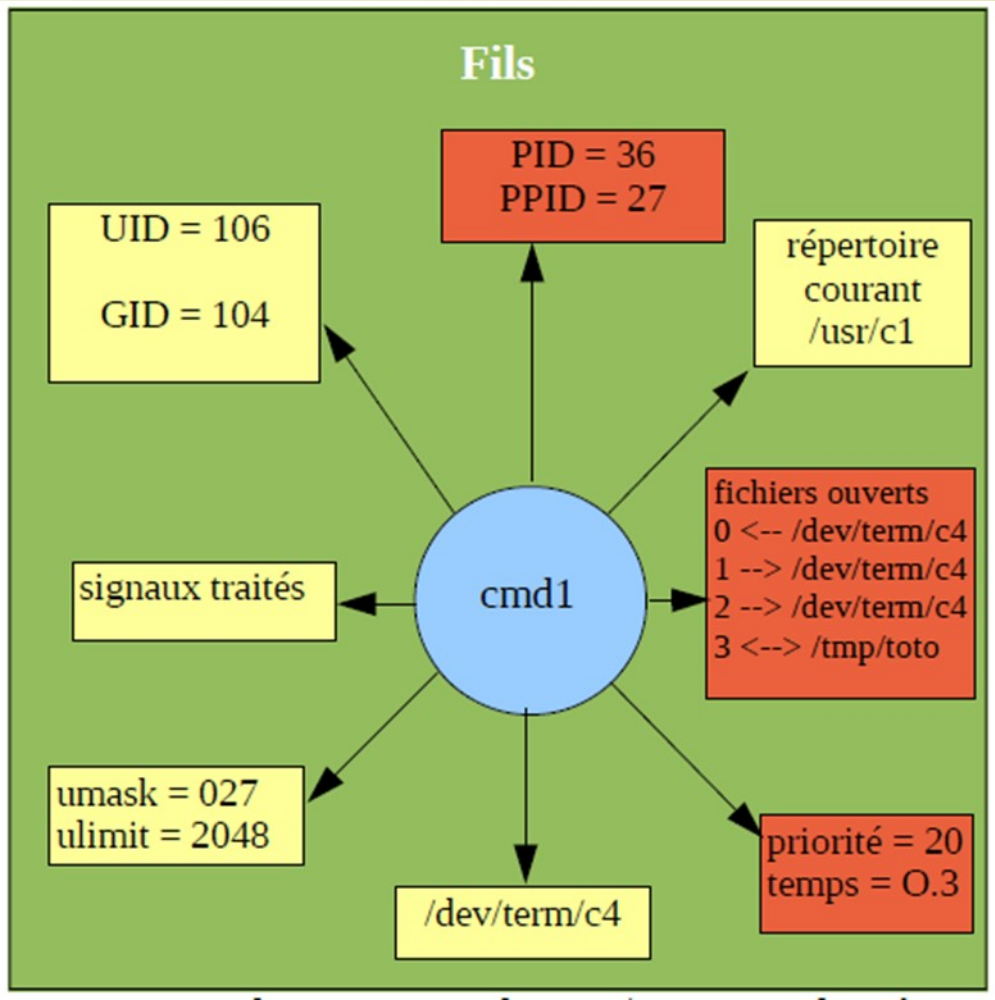
Le fils hérite de tout l'environnement du père, sauf bien sûr du PID, du PPID et des temps d'exécution.



Création de processus :

Un nouvel élément de l'environnement apparaît ici, le PPID. C'est le PID du processus père. Le père du processus 36 est le processus 27, et celui de 27 est le processus 1. Seul le fils (36) a ouvert le fichier /tmp/toto. Les deux processus peuvent parfaitement tourner en parallèle. La puissance de traitement est partagée entre tous les programmes lancés et, mis à part les machines multi- processeurs, un seul processus est actif à un instant t.

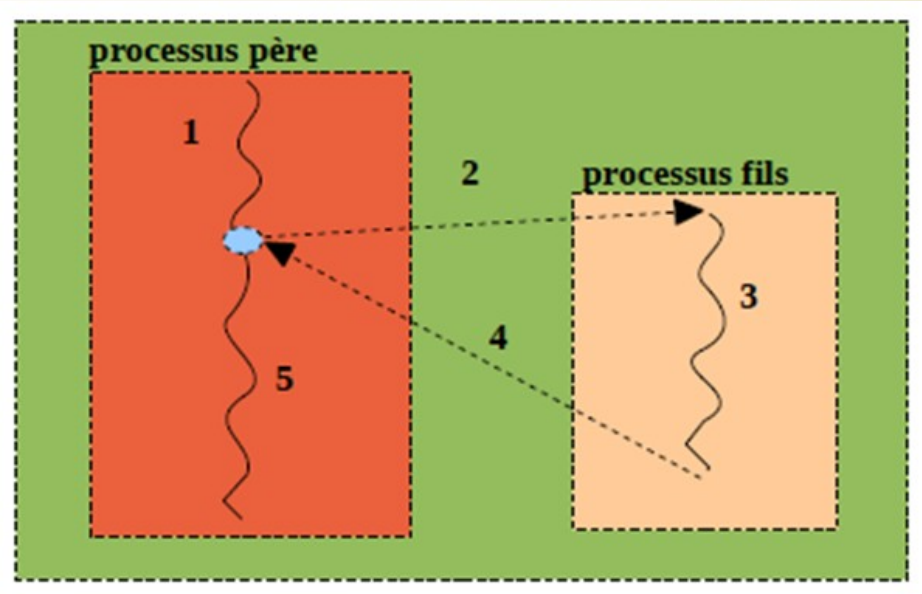




Enchainement de processus :

On utilisera cette solution (processus lancés en parallèle) par exemple pour lancer un traitement très long, et continuer à travailler en même temps. Dans ce cas, on dit que le père a lancé un fils en tâche de fond (background) ou encore en mode asynchrone.

Une autre solution consiste à placer le processus père en attente jusqu'à ce que le processus fils soit terminé.



Pour lancer une commande en plaçant le père en attente, il suffit de taper la commande :

prompt> cmd1
... résultat de la commande cmd1
prompt>

Ce mode est donc le mode par défaut dans le Shell.

Tâche de fond :

Pour lancer une commande en tâche de fond, il faut faire suivre cette commande par le caractère '**&**':

```
prompt> cmd1 &
 [1] 127
prompt>
```

Le Shell affiche un numéro de tâche entre « [ ] » et le PID de cette tâche de fond, puis continue à travailler, donc affiche la chaîne d'invite et attend la prochaine commande.

En Bourne Shell, il n'y a pas de numéro de tâche, la même commande aurait donnée :


```
prompt> cmd1 &
127
prompt>
```

Si vous avez plusieurs commandes successives à lancer en arrière plan, il faudra utiliser les parenthèses.

```
prompt> (cmd1; cmd2) &
 [2] 128
prompt>
```

La commande **cmd2** ne sera lancée que lorsque la commande **cmd1** sera terminée. Ceci dit, l'utilisateur récupère la main tout de suite. Le Shell détecte la présence du '**&**' partout sur la ligne.

Tâche de fond :

Dans le cas suivant, la commande cmd1 sera lancée par un sous-processus shell :

```
prompt> (cmd1)
prompt>
```

Dans le cas suivant, la commande **cmd1** est lancée en arrière plan et la commande **cmd2** est tout de suite lancée derrière, en direct (en parallèle).

```
prompt> cmd1 & cmd2
 [3] 130
prompt>
```


La commande « wait n » permet d'attendre la mort de la tâche de fond dont le PID est « n ». .

```
prompt> cmd1 &
 [4] 132
prompt> wait 132
... on est bloqué jusqu'à ce que cmd1 se termine
prompt>
```

Si « n » n'est pas précisée, wait attend la mort de toutes les tâches de fond. wait ne s'applique qu'aux processus lancés dans le shell lui-même..

Arborescence de processus :

Tous les processus sont créés à partir d'un processus père, existant déjà.

Le premier processus est un peu spécial. Il est créé lorsque le système est initialisé. Il s'appelle "**init**", a le **PID 1** et n'est associé à aucun terminal. Son travail consiste à créer de nouveaux processus.

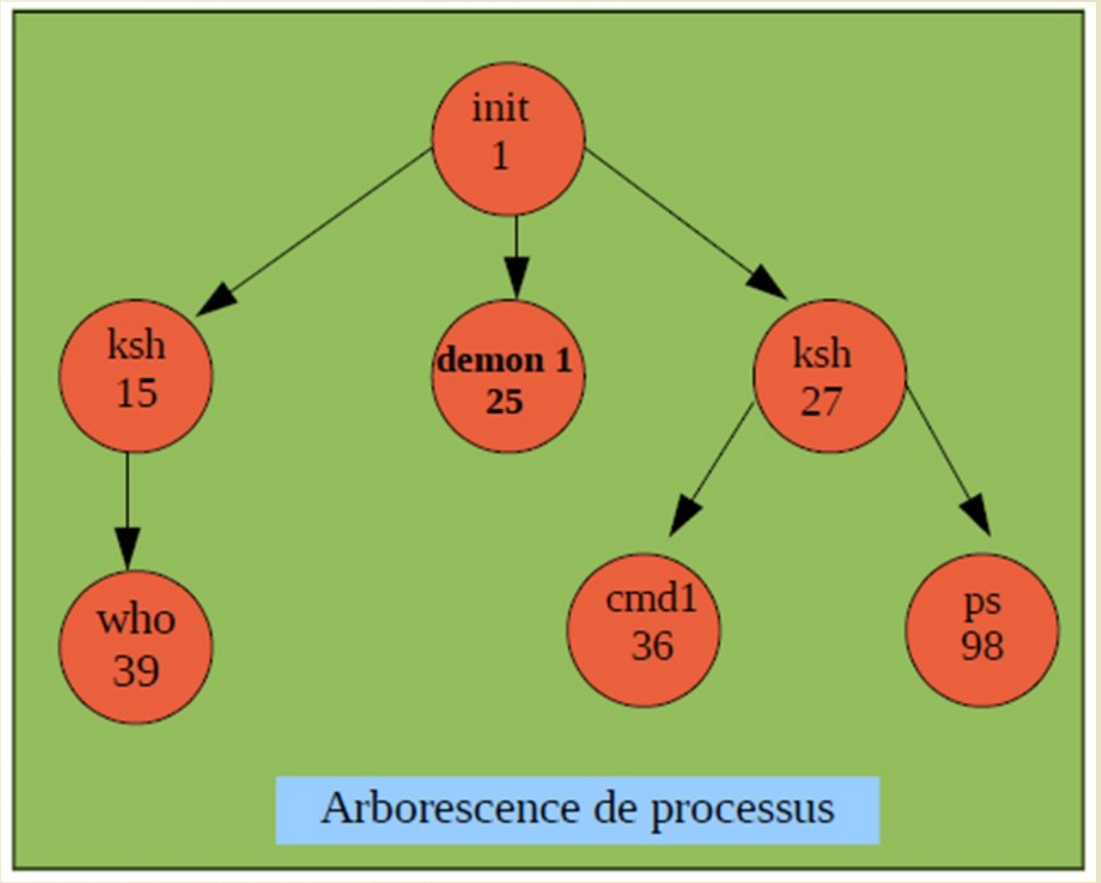
Le processus "**init**" crée 2 sortes de processus :

- des **démons**, c'est-à-dire des processus qui ne sont rattachés à aucun terminal, qui sont **endormis** la plupart du temps, mais qui se **réveillent** de temps en temps pour effectuer une tâche précise (par exemple la gestion des imprimantes).
- **des processus interactifs**, associés aux lignes d'entrées/sorties sur lesquelles sont rattachés des terminaux. Autrement dit des processus vous permettant de vous connecter.

Arborescence de processus :

Pour visualiser les processus que vous avez lancé, tapez la commande «**ps**» :

```
prompt> echo $$
527
prompt> cmd1 &
prompt>
prompt> ps
PID TTY TIME COMMAND
527 ttyp4 1:70 -ksh
536 ttyp4 0:30 cmd1
559 ttyp4 0:00 ps
prompt>
```



- PID identifie le processus,
- TTY est le numéro du terminal associé,
- TIME est le temps cumulé d'exécution du processus,
- COMMAND est le nom du fichier correspondant au programme exécuté par le processus.

La commande ps :

Sans option, la commande concerne les processus associés au terminal depuis lequel elle est lancée.

- ps -ef # liste de tous les processus
- ps -ef | grep firefox # Firefox est-il actif ?
- ps aux # afficher les ressources utilisées
- ps -u root # les processus associés à un UID

Il existe bien d'autres commandes pour gérer les processus,

comme par exemple la commande « top ».

La commande top :

Cette commande affiche en temps réel les processus qui consomment le plus de ressources systèmes. Dans les premières lignes, elle affiche des informations globales sur le système (charge, mémoire, nombre de processus, ...).

```
top - 11:14:18 up  1:02,  1 user,  load average: 0,05, 0,09, 0,08
Tasks: 209 total,   1 running, 208 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,1 us,  0,2 sy,  0,0 ni, 99,8 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Mem :   7933,1 total,   6381,5 free,    797,0 used,    754,6 buff/cache
MiB Swap:   2048,0 total,   2048,0 free,      0,0 used.   6866,3 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    607 systemd+  20   0   16004   6300   5472 S   0,3   0,1   0:03.81 systemd+
   1327 jfa       20   0 4053620 269836 129020 S   0,3   3,3   0:37.19 gnome-s+
   2104 jfa       20   0  227344   2432   2072 S   0,3   0,0   0:06.34 VBoxCli+
   2207 jfa       20   0  563848  54248  41684 S   0,3   0,7   0:02.81 gnome-t+
   2718 jfa       20   0 2819336  64620  49212 S   0,3   0,8   0:00.68 gjs
      1 root      20   0  167916  12340   8576 S   0,0   0,2   0:01.75 systemd
      2 root      20   0       0      0      0 S   0,0   0,0   0:00.00 kthreadd
      3 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_par+
      5 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 netns
      6 root      20   0       0      0      0 I   0,0   0,0   0:01.01 kworker+
      7 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 kworker+
      9 root       0 -20       0      0      0 I   0,0   0,0   0:00.06 kworker+
     10 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 mm_perc+
     11 root      20   0       0      0      0 I   0,0   0,0   0:00.00 rcu_tas+
     12 root      20   0       0      0      0 I   0,0   0,0   0:00.00 rcu_tas+
     13 root      20   0       0      0      0 I   0,0   0,0   0:00.00 rcu_tas+
```

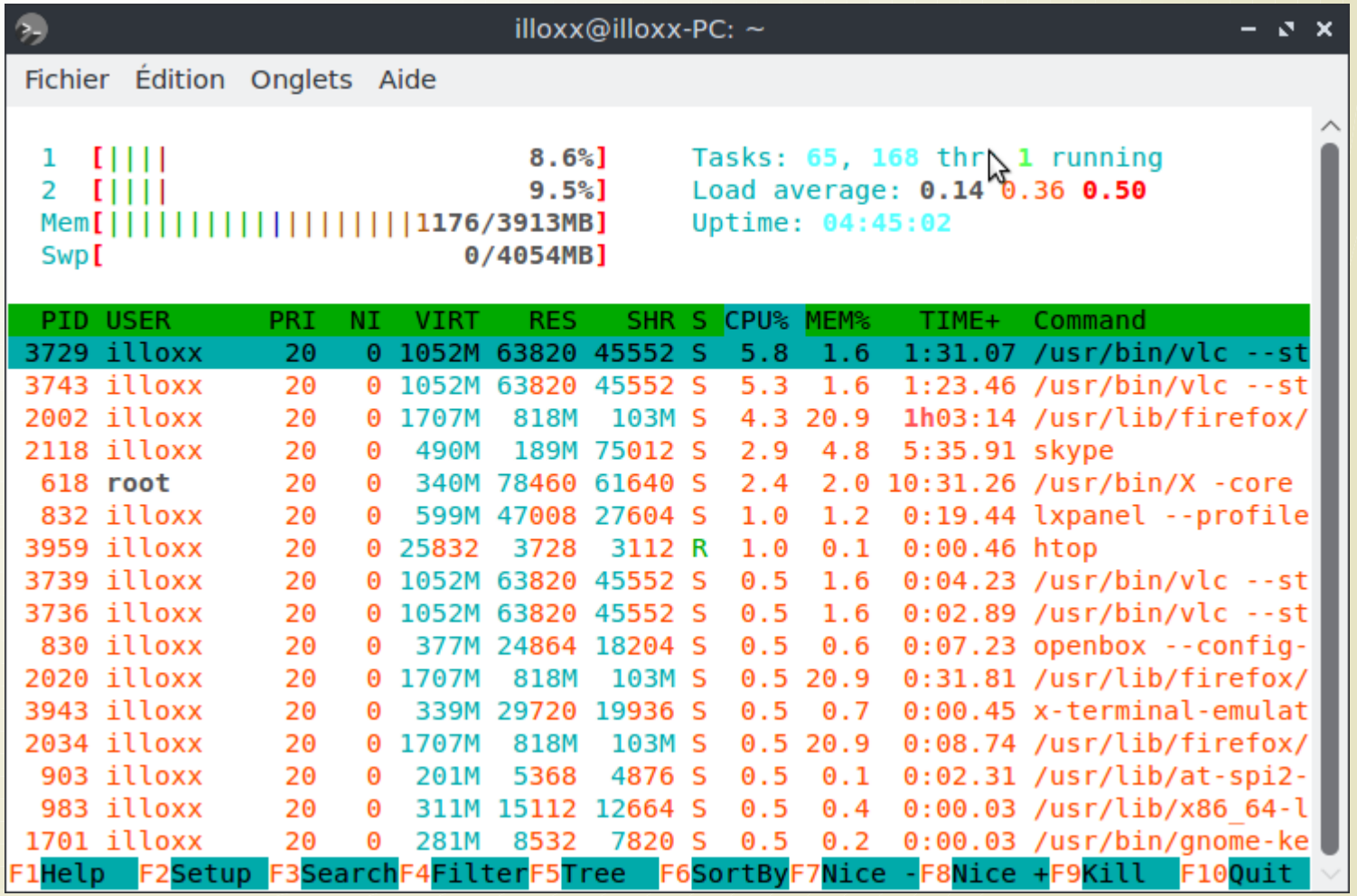
https://manpages.ubuntu.com/manpages/xenial/fr/man1/top.1.html

La commande htop :

HTOP est un outil logiciel à utiliser en ligne de commande.

```
sudo apt install htop
```

Il permet de superviser les ressources sur un serveur.



https://doc.ubuntu-fr.org/htop

Retour de processus :

Lorsqu'un processus se termine, il retourne toujours une valeur significative ou statut.

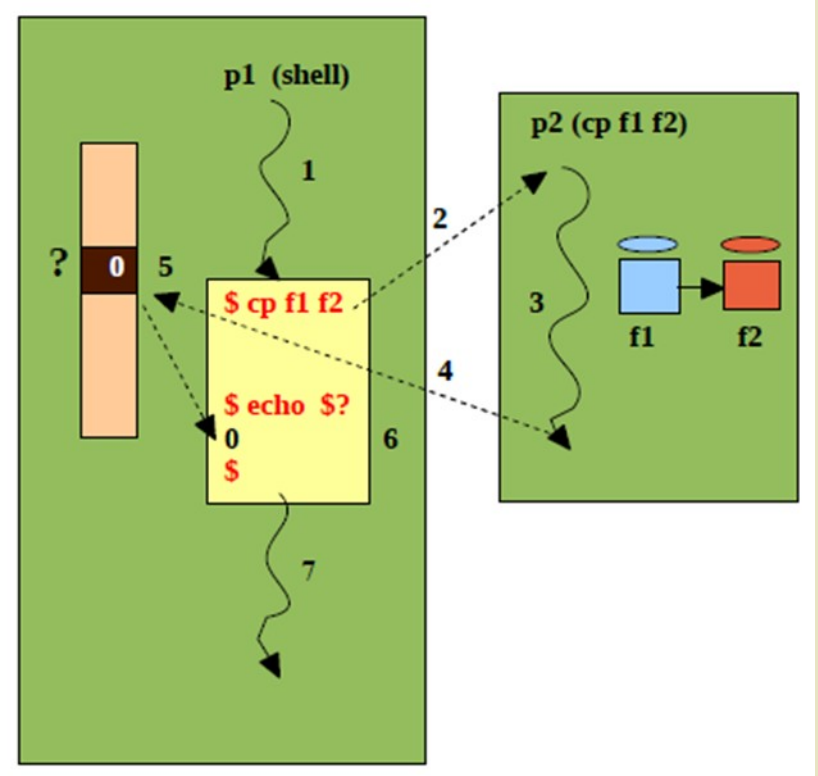
Par convention, lorsqu'un processus se termine correctement, il retourne la valeur **0**, sinon il retourne une valeur différente de **0** (généralement **1**). Ce choix permet de ramener des codes significatifs pour différencier les erreurs.

Le statut d'une commande Shell est placé dans la pseudo-variable spéciale, nommée « **?** ». On peut consulter sa valeur en tapant la commande :

```
echo $?
```


Exemple :

```
prompt> cp f1 f2
prompt> echo $?
0
Prompt>
```



Sur cet exemple, la valeur 0 renvoyée par la commande « **echo** » nous indique que la commande « **cp** » s'est bien passée.

Autres commandes de gestion de processus :

La commande kill

kill envoie un signal à un (des) processus .

Syntaxe générale

```
kill [option] [PID]
```

La commande **kill** envoie un signal à des processus ou groupes de processus spécifiés, les obligeant à agir en fonction du signal. Lorsque le signal n'est pas spécifié, il est défini par défaut sur -15 (-TERM). The most commonly used signals are :

- 1 (HUP) - Reload a process.
- 9 (KILL) - Kill a process.
- 15 (TERM) - Gracefully stop a process.
- Pour obtenir une liste de tous les signaux disponibles, appelez la commande avec l'option -l

Exemple :

```
$ ps
    PID TTY          TIME CMD
   2226 pts/0    00:00:00 bash
   3614 pts/0    00:00:00 vi
   3617 pts/0    00:00:00 ps
$ kill -15 3614
```

```
$ ps
    PID TTY          TIME CMD
   2226 pts/0    00:00:00 bash
   3635 pts/0    00:00:00 ps
[1]+  Killed  vi toto
$
```

Autres commandes de gestion de processus :

La commande killall

killall idem à kill mais pour tous les processus qui exécutent une commande spécifique .

Syntaxe générale

```
killall [option] Processus
```

La commande **killall** envoie un signal à tous les processus ou groupes de processus dont le nom est spécifié, les obligeant à agir en fonction du signal. Lorsque le signal n'est pas spécifié, il est défini par défaut sur -15 (-TERM).

The most commonly used signals are :

- 1 (HUP) - Reload a process.
- 9 (KILL) - Kill a process.
- 15 (TERM) - Gracefully stop a process.
- Pour obtenir une liste de tous les signaux disponibles, appelez la commande avec l'option -l

Autres commandes de gestion de processus :

La commande jobs

jobs est une commande des systèmes d'exploitation Unix et Unix-like pour lister les processus lancés ou suspendus en arrière-plan.

Syntaxe générale

```
jobs [option] [jobID]
```

La commande **jobs** liste es processus en cours d'exécution ainsi que leur état : running ou stopped ou done,

Exemple :

```
$ nano f1 &
$ firefox &
$ jobs
[1]-  Stopped                 nano f1
[2]+  Running                 firefox &
$
```

Autres commandes de gestion de processus :

La commande fg

fg est la commande qui permet de remettre un processus au premier plan (foreground)

Syntaxe générale

```
fg [options] %[jobID]
```

Le **jobID** est le numéro fournit par la commande **jobs**

Exemple :

```
$ jobs
[1]-  Stopped                 nano f1
[2]+  Running                 firefox &
$ fg %2
Affiche la fenêtre Firefox !
```

Autres commandes de gestion de processus :

La commande bg

bg est la commande qui permet de remettre un processus au arrière plan (background)

Syntaxe générale

```
bg [options] %[jobID]
```

Le **jobID** est le numéro fournit par la commande **jobs**

Exemple :

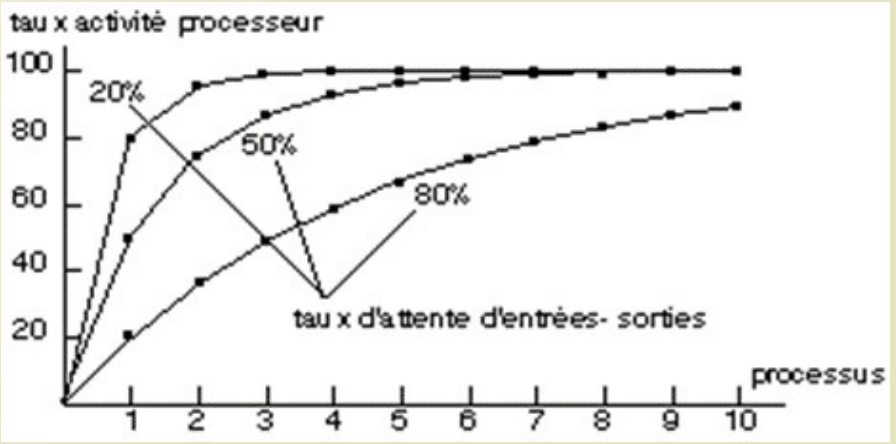
```
$ jobs
[1]-  Stopped                 nano f1
[2]+  Running                 firefox &
$ bg %2
Remet la fenêtre Firefox en arrière plan !
```

Multi programmation :

Les processus correspondent donc à l'exécution de travaux de la part du système d'exploitation : les programmes des utilisateurs, la gestion des entrées-sorties..., les tâches du kernel. La plupart des systèmes d'exploitation nous donnent l'impression qu'ils sont en mesure de faire tourner plusieurs processus simultanément.

En fait, l'ordinateur à un seul processeur résout ce problème grâce à un pseudo-parallélisme. Il traite une tâche à la fois, s'interrompt et passe à la suivante. La commutation des tâches étant très rapide, l'ordinateur donne **l'illusion** d'effectuer un traitement simultané. Cette capacité de faire tourner plusieurs tâches à la fois se nomme *multiprogrammation* (ou multiplexage) et est au cœur des systèmes d'exploitation modernes.

Elle permet notamment de maximiser l'utilisation de processeur. Par exemple, dans la figure ci-contre, si l'on a un seul processus qui tourne sur une machine donnée, le taux d'utilisation du processeur n'est que de 20% lorsque le taux d'attente d'entrées-sorties est de 80 % (courbe du bas). Par contre, pour la même courbe, on passe à 80 % d'utilisation lorsqu'on augmente à 8 le nombre de processus qui peuvent tourner « en même temps ».



La figure suivante montre plusieurs processus en mémoire mais un seul à la fois reçoit du temps de CPU.

http://deptinfo.cnam.fr/Enseignement/CycleA/AMSI/exercices_systemes/physiq15.htm

JFA 192

Mauvais exemple :

```
$ find /usr/c1 -name *.c -print
```

Dans la commande **find**, le shell remplace **.c* par la liste des fichiers finissant par *.c* du répertoire */usr/c1*, puis va chercher dans l'arborescence donnée ces noms de fichiers.

Cela reviendra à :

```
$ find /usr/c1 -name f1.c f2.c f3.c -print
```

Seulement **-name** n'accepte qu'un seul argument !

Par contre dans l'expression suivante :

```
$ find /usr/c1 -name '*'.c -print
```

C'est bien le modèle ****.c*** qui sera passé en argument de l'option **-name** de la commande **find**. La recherche se fera donc bien sur les trois fichiers ***f1.c*, *f2.c* et *f3.c.***

- ***Le critère de recherche perm***

```
-perm nombre_octal sélectionne les fichiers dont les droits d'accès sont ceux indiqués par le nombre octal.
```

Exemple :

```
$ find /usr/c1 -perm 777 -print
```

Affiche tous les fichiers qui sont autorisés en lecture, écriture et exécution pour l'utilisateur propriétaire, les personnes du groupe propriétaire et tous les autres.

- ***Le critère de recherche type***

```
-type caractère sélectionne les fichiers dont le type est celui indiqué par le caractère.
```


C'est-à-dire :

c pour un fichier spécial en mode caractère

b pour un fichier spécial en mode bloc

d pour un répertoire

f pour un fichier normal

l pour un lien symbolique

. ***Exemple :***

```
$ find /usr/c1 -type d -print
```

Cette commande affiche tous les répertoires et sous- répertoires de ***/usr/c1***.

- ***Le critère de recherche links***

-links *nombre_décimal* sélectionne les fichiers dont le nombre de liens est donné par le nombre décimal. Si le nombre est précédé d'un + (d'un -) cela signifie supérieur (inférieur) à ce nombre.

Exemple :

```
$ find /usr/c1 -links +2 -print
```

Cette commande affiche tous les fichiers qui ont plus de deux liens.

- ***Le critère de recherche user***

-user *n[ou]m_utilisateur* sélectionne les fichiers dont l'utilisateur propriétaire est ***nom_utilisateur*** ou dont le numéro d'utilisateur (UID) est ***num_utilisateur***.

Exemple :

```
$ find /dev -user c1 -print
```

Cette commande affiche tous les fichiers spéciaux appartenant à l'utilisateur ***c1***.

- ***Autres critères de recherche :***

-inum *nombre_décimal :* sélectionne les fichiers ayant pour numéro d'i- noeud ***nombre_décimal***.

-newer *fichier* sélectionne les fichiers qui sont plus récents que celui passé en argument.

-atime *nombre_décimal :* sélectionne les fichiers qui ont été accédés dans les ***nombre_décimal*** derniers jours.

-mtime *nombre_décimal :* sélectionne les fichiers qui ont été modifiés dans les ***nombre_décimal*** derniers jours.

-size *nombre_décimal [c] :* sélectionne les fichiers dont la taille est de ***nombre_décimal*** blocs. Si on post-fixe le ***nombre_décimal*** par le caractère **c**, alors la taille sera donnée en nombre de caractères.

- ***Combinaison de critères :***

Plusieurs de ces critères peuvent être groupés par les opérateurs (et).

Attention : "**(**" et "**)**" étant des caractères spéciaux pour le shell, ils doivent être protégés.

Si plusieurs critères sont mis à la suite, **find** sélectionne les fichiers qui répondent à tous les critères. Le "ET logique" est donc implicite.

Exemple :

```
$ find /usr/c1 \( -name '*.c' -mtime -3 \) -print
```


Cette commande affiche les fichiers se terminant par "**.c**" et modifiés dans les 3 derniers jours.

- ***Autres combinaisons de critères de recherche :***

Le "OU logique" est représenté par l'opérateur -o

Exemple :

```
$ find /usr/c1 \( -name '*.txt' -o -name '*.doc' \) -print
```

Cette commande affiche tous les fichiers se terminant par "**.txt**" ou "**.doc**".

Le "NON logique" est l'opérateur !

Exemple :

```
$ find /usr/c1 ! -user c1 -print
```

Cette commande affiche tous les fichiers n'appartenant pas à **c1**, mais qui se trouvent dans son arborescence.

Les actions possibles sur les noms de fichiers sélectionnés

-print affiche le nom des fichiers sélectionnés sur la sortie standard.

Exemple :

```
$ find /usr/c1 -print
```

Cette commande affiche toute l'arborescence de ***c1***.

-exec *commande* \; exécute ***commande*** sur tous les fichiers sélectionnés. Dans la commande shell, "**{}**" sera remplacé par le nom du fichier sélectionné.

Exemples :

```
$ find /usr/c1 -name '*.o' -exec rm {} \;
$ find /usr/c1 -name '*.o' -print
```

Cette commande recherche tous les fichiers se terminant par l'extension "**.o**" dans l'arborescence "**/usr/c1**" et les détruit, puis recherche les fichiers se terminant par l'extension "**.o**" pour vérifier

```
$ find /dev /home -user hugo -exec ls -l {} \;
```

Cette commande affiche tous les fichiers appartenant à hugo, en format "long". La recherche est effectuée dans les répertoires **/dev** et **/home**.

La commande Type :

Type :

```
type commande ...
```

Donne le chemin absolu du fichier exécuté lorsque vous tapez *commande*. Sinon, indique que la commande est interne au shell.

Exemple :

```
prompt> type find pg
find is /bin/find
pg is /usr/bin/pg

prompt> type umask
umask is a shell builtin
umask est une primitive du shell
prompt>
```

La commande File :

```
file fichier
```

affiche le type du fichier (exécutable, répertoire, ASCII ...). En l'utilisant avant de visualiser le contenu d'un fichier cela évite d'afficher un contenu binaire.

Exemple :

```
prompt> file /bin/ls /etc/passwd /usr/c1
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.15, stripped
/bin/ls: demand paged pure executable

/etc/passwd: ASCII text

/usr/c1: directory

prompt>
```

La commande Test :

`test expression ou [ expression ]`

test évalue ***expression*** et retourne le résultat de cette évaluation. **test** appelé sans argument retourne ***faux***.

Vous devez utiliser une des deux syntaxes, mais pas les deux en même temps ! Il ne faut pas non plus oublier de mettre les caractères séparateurs (blanc, tabulation ...) entre les caractères [et].

Attention aux nombres d'arguments ! En effet la commande **test** prend un nombre d'arguments bien spécifique ; lorsque l'on fait des tests avec des variables, il vaut mieux les encadrer de double quotes ( ). Dans ce cas, si la variable n'a pas de valeur, alors la commande **test** trouvera un mot vide (mais existant).

Test sur des fichiers et répertoires

- test -**w** *fichier :* vrai si ***fichier*** existe et est autorisé en **écriture**.
- test -**r** *fichier :* vrai si ***fichier*** existe et est autorisé en **lecture**.
- test -**x** *fichier :* vrai si ***fichier*** existe et est **exécutable**. test -**d** *fichier* vrai si ***fichier*** existe et est un **répertoire**.
- test -**f** *fichier :* vrai si ***fichier*** existe et n'est **pas un répertoire**.
- test -**s** *fichier :* vrai si ***fichier*** existe et a une **taille non nulle**.
- test -**t** *descripteur :* vrai si ***descripteur*** est associé à un **terminal**.

Exemples :

```
prompt> ls -l
drwxr-x--- 11 c1 cours 17 Aug 1 09:00 save
-rw-r----- 1 fun axis 21 Jul 25 17:05 data

prompt> who am i
c1 term/c4 Aug 2 09:01

prompt> test -f save; echo $?
1
prompt> test -d save; echo $?
0
prompt> test -r save; echo $?
0
prompt> test -f data; echo $?
0
prompt> test -w data; echo $?
1
prompt>
```

Test sur des chaînes

- test -**z** *s1 :* vrai si la chaîne ***s1*** est **vide** (a une longueur de 0 caractère).
- test -**n** *s1 :* vrai si la chaîne ***s1*** est **non vide**.
- test *s1 = s2 :* vrai si les chaînes ***s1*** et ***s2*** sont **identiques**. test *s1* != *s2* vrai si les chaînes ***s1*** et ***s2*** sont **différentes**.
- test *s1 :* vrai si la chaîne ***s1*** n'est **pas la chaîne nulle**.

Test sur des nombres

- test *n1* **-eq** *n2 :* vrai si l'entier ***n1*** est **égal** à l'entier ***n2***. test *n1* **-ne** *n2* vrai si l'entier ***n1*** est **différent** de l'entier ***n2***.
- test *n1* **-gt** *n2* vrai si l'entier ***n1*** est **supérieur** à l'entier ***n2***.
- test *n1* **-lt** *n2* vrai si l'entier ***n1*** est **inférieur** à l'entier ***n2***.
- test *n1* **-ge** *n2* vrai si l'entier ***n1*** est **supérieur ou égal** à l'entier ***n2***.
- test *n1* **-le** *n2* vrai si l'entier ***n1*** est **inférieur ou égal** à l'entier ***n2***.

Exemple :

```
$ cat testarg
if test "$#" -eq "0" then
echo "usage: $0 arg1 arg2" >&2 exit 1
fi
$
```

Si le nombre de paramètres passés à l'appel de la commande vaut 0, on quitte la commande avec un message d'erreur sur la sortie standard (sortie 1 recopiée sur la sortie 2).

PS : entre simples quotes le shell n'interprète pas les variables …

```
prompt> echo ' usage : $0 arg1 arg2 '
usage : $0 arg1 arg2
prompt>
```

L'opérateur ET logique

```
cmd1 && cmd2
```

On exécutera ***cmd2*** uniquement si la commande ***cmd1*** se termine correctement

Exemple :

```
$ pwd
/usr/c1
$ mkdir tmp
$ test -d $HOME/tmp && cd $HOME/tmp
$ pwd
/usr/c1/tmp
$ cd
$ rmdir tmp
$ test -d $HOME/tmp && cd $HOME/tmp
$ pwd
/usr/c1
$
```

S'il existe un répertoire ***tmp*** dans le répertoire courant, alors aller dans ce répertoire.

L'opérateur ET logique

On pourrait obtenir le même résultat en utilisant la structure de contrôle ***if ... then ...fi*** :

```
$ pwd
/usr/c1
$ mkdir tmp
$ if test -d $HOME/tmp
> then cd tmp
> fi
$ pwd
/usr/c1/tmp
$ cd
$ rmdir tmp
$ if test -d $HOME/tmp
> then cd tmp
> fi
$ pwd
/usr/c1
$
```

L'opérateur OU logique

```
cmd1 || cmd2
```

On exécutera ***cmd2*** uniquement si la commande ***cmd1*** ne se termine pas correctement (avec un statut différent de 0). Le statut enfin retourné est celui de la dernière commande exécutée.

Exemple :

```
$ pwd
/usr/c1
$ mkdir tmp
$ test -d $HOME/tmp || echo $HOME/tmp inexistant
$ rmdir tmp
$ test -d $HOME/tmp || echo $HOME/tmp inexistant
/usr/c1/tmp inexistant
$
```

S'il n'existe pas de répertoire ***tmp*** dans le répertoire courant, alors afficher un message.

L'opérateur ET logique

De la même façon que pour l'opérateur logique ET, on pourrait ici obtenir le même résultat en utilisant la structure de contrôle ***if ... then ... fi*** *:*

```
$ pwd
/usr/c1
$ mkdir tmp
$ if test ! -d $HOME/tmp
> then
> echo $HOME/tmp inexistant
> fi
$ rmdir tmp
$ if test ! -d $HOME/tmp
> then
> echo $HOME/tmp inexistant
> fi
/usr/c1/tmp inexistant
$
```

Combinaison de primitives

On peut combiner toutes ces primitives avec les opérateurs :

- ***!*** négation
- ***-a*** ET logique
- ***-o*** OU logique
- ***(*** *expression* ***)*** pour regrouper logiquement plusieurs tests.

Remarques :

L'opérateur **-a** est plus prioritaire que **-o**.

Chaque primitive, opérateur ou opérande doit constituer un mot pour le shell, donc doit être séparé par des blancs.

Les parenthèses doivent être protégées (par un ***backslash*** ou encadrées par des quotes) pour éviter que le shell ne les interprète.

Exemple :

```
$ cat chem

if test -d "$1" -a -x "$1" then
echo chemin accessible
cd $1
else
echo chemin inaccessible
fi
$
```


Si la valeur du premier argument est un répertoire et que l'on est autorisé à se déplacer dedans(-x) , alors on y va. Sinon on affiche un message.

La commande expr

`expr arg ...`

L'expression formée par les arguments, est évaluée, et **expr** retourne le résultat de cette évaluation.

Attention : certains opérateurs utilisés par la commande **expr** sont significatifs pour le shell. Ils devront donc être protégés.

Exemple :

- pour **|** utiliser : **'|'** ou encore "|" ou encore **\|**
- pour **>=** utiliser : **'>='** ou encore **">="** ou encore **\>=**
- pour ***** utiliser : **'*'** ou encore **"*"** ou encore *****

Plusieurs arguments peuvent être combinés à l'aide des opérateurs suivants :

- e1 | e2 si ***e1*** est égale à 0 retourne ***e2*** sinon retourne ***e1***
- e1 & e2 si ni ***e1*** ni ***e2*** ne sont égales à 0 retourne ***e1***
- e1 < e2 retourne 1 si ***e1*** est plus petit que ***e2*** sinon 0
- e1 <= e2 retourne 1 si ***e1*** est plus petit ou égal ***e2*** à sinon 0
- e1 = e2 retourne 1 si ***e1*** est égal à ***e2*** sinon 0
- e1 != e2 retourne 1 si ***e1*** est différent de ***e2*** sinon 0
- e1 > e2 retourne 1 si ***e1*** est égal à ***e2*** sinon 0
- e1 >= e2 retourne 1 si ***e1*** est supérieur ou égale ***e2*** à sinon 0

Si les deux expressions sont numériques, la comparaison sera numérique, sinon elle sera lexicographique.

- e1 + e2 retourne le résultat de l'addition
- e1 - e2 retourne le résultat de la soustraction
- e1 * e2 retourne le résultat de la multiplication
- e1 / e2 retourne le résultat de la division
- e1 % e2 retourne le résultat du modulo

Exemple :


```
a=3
b=$(expr $a + 5)
```

La variable "**b**" va récupérer le résultat du calcul effectué entre parenthèses, c'est-à-dire "**8**". La variable "**a**" est inchangée.

Il existe une notation simplifiée :

```
a=3
b=$(( $a + 5))
```

La commande for

Bash a introduit une nouvelle structure *for* adaptée aux traitements des expressions arithmétiques, itération issue du langage C.

Elle fonctionne comme cette dernière.

```
for (( expr_arith1 ; expr_arith2 ; expr_arith3 ))
do suite_cmd done
```

- *expr_arith1* est l'expression arithmétique d'initialisation.
- *expr_arith2* est la condition d'arrêt de l'itération.
- *expr_arith3* est l'expression arithmétique qui fixe le pas d'incrémentation ou de décrémentation.

Exemples :

```
declare -i x
for (( x=0 ; x<5 ; x++ ))
do
echo $(( x*2 ))
done

declare -i x y
for (( x=1,y=10 ; x<4 ; x++,y-- ))
do
echo $(( x*y ))
done
```

Webographie

- https://icon-icons.com/fr/icone/
- https://www.courstechinfo.be/Techno/Historique2.html
- http://e-classroom.over-blog.com/les-systemes-d-exploitation
- https://distrowatch.com/
- https://www.editions-eni.fr